FPGA ACCELERATED DISCRETE-SURF FOR REAL-TIME HOMOGRAPHY ESTIMATION

THESIS

Andrew C. Leighner, Second Lieutenant, USAF

AFIT-ENG-MS-15-M-042

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-15-M-042

FPGA ACCELERATED DISCRETE-SURF FOR REAL-TIME HOMOGRAPHY ESTIMATION

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Engineering

Andrew C. Leighner, B.S.C.E.
Second Lieutenant, USAF

March 2015

AFIT-ENG-MS-15-M-042

FPGA ACCELERATED DISCRETE-SURF FOR REAL-TIME HOMOGRAPHY ESTIMATION

Andrew C. Leighner, B.S.C.E.
Second Lieutenant, USAF

Committee Membership:

Maj John Pecarina, PhD
Chair

Maj Brian Woolley, PhD
Co-Chair

Dr. Gilbert Peterson
Member

AFIT-ENG-MS-15-M-042

Abstract

This paper describes our hardware accelerated, FPGA implementation of SURF, named Discrete SURF, to support real-time homography estimation for close range aerial navigation. The SURF algorithm provides feature matches between a model and a scene which can be used to find the transformation between the camera and the model. Previous implementations of SURF have partially employed FPGAs to accelerate the feature detection stage of upright only image comparisons. We extend the work of previous implementations by providing an FPGA implementation that allows rotation during image comparisons in order to facilitate aerial navigation. We also expand beyond feature detection as the complete Discrete SURF algorithm is run on the FPGA, rather than piped into processors. This not only minimizes overhead and increases the parallelization of the algorithm, but also allows the algorithm to be easily ported to different FPGAs. Furthermore, the Discrete SURF module is a logic-only implementation that does not rely on external hardware which therefore decreases the overall size, weight and power of the device while also allowing for easy FPGA to ASIC conversion. We evaluate the Discrete SURF algorithm in terms of performance against the original SURF and upright SURF algorithms implemented in OpenCV. Finally, we show how Discrete SURF is more compatible with an aerial navigation scenario than previous works, since rotation invariance must be considered in addition to scale.

Table of Contents

Page

List of Figures

List of Tables

FPGA ACCELERATED DISCRETE-SURF FOR REAL-TIME HOMOGRAPHY ESTIMATION

I. Introduction

Recent years have shown a dramatic increase in the use and interest in unmanned aerial vehicles (UAVs) stemming from proven effectiveness in military use and proceeding to corporate venture (i.e. Amazon)[20]. With the proliferation of UAV technology, new research areas are emerging in swarm persistence[23][28], perching or aerial gripping[10], battery swapping[1][27], autonomous air refueling[18] and landing on unique surfaces[8] and moving platforms[12][29]. Underlying these new research areas is a need for methods to assist navigation in close proximity to a target. The immediate concern is for the availability of specialized navigation algorithms implemented on UAV compatible hardware

1.1 Close Range Navigation

The autonomous approach of a UAV to a target platform requires real-time position knowledge to protect the UAV against sudden, unanticipated movement caused by wind, weather or another airborne object (especially autonomous ones). An estimator and flight controller system must be accurate enough at close distances to avoid collisions and allow for the precision needed to carry out air-to-air tasks. Thus, evaluating the effectiveness of close range navigation mechanisms in real flight tests could be potentially hazardous and costly. To support a near term research agenda, it is prudent to constrain implementation to low cost devices and hardware. Therefore, we constrain the type of UAV compatible hardware to low size, weight, and power

(SWAP) platforms for quad-rotor and mini-UAV mounting.

Existing solutions to autonomous navigation provide varying degrees of accuracy but have different environmental and preexisting knowledge requirements. GPS and inertial systems are the most common means of autonomous navigation, but both are limited in that they are only capable of navigation in known environments or around obstacles that communicate their position[17]. Sonar, radar, and lidar systems are also common approaches to navigation that are effective means of determining distance information, but can take time to build an environment map, and their size, weight and power (SWAP) properties often make them impractical for real-time operation on a small UAV.

1.2 Vision-Based Navigation

Vision navigation methods provide a resonable accurate detection capability, but are limited in detection distance by the resolution of the images taken[13]. Fortunately, even small, inexpensive cameras can provide the resolution need for close range navigation[19]. Additionally, feature matching algorithms require preloaded knowledge of environment objects to identify them, but we assume that it is easy to make these models available at configuration. Furthermore, cameras can quickly capture large amounts of environment data in frame, however, the problem lies in accurately interpreting position and orientation data from the frame in real-time.

A number of algorithms that offer scale and rotation invariant feature tracking have emerged that characterize raw image data into usable environment points. These features can be compared between a scene image and a model image at a known orientation to find a homography transform. The reader is referred to the seminal works that provide feature matches needed for robust homography estimation from a single image[4][14]. The homography matrix provides the information needed to determine the orientation and rotation transformation between the camera and the target object. In short, the homography estimation system must be accurate enough at close distances to avoid collisions and allow for the precision needed to carry out air-to-air tasks.

Outside of the aerial navigation field, the SIFT and SURF algorithms have been sped up to the point of real-time image processing by implementing them on specialized hardware. Both SURF and SIFT have been implemented in GPUs, which has dramatically increase the processing speed, as feature tracking algorithms are highly parallelizable by their nature[25][30]. For the same reason, field programmable gate arrays (FPGAs) are also being explored as a platform for vision based position estimation with lower power requirements. To date, there have not been any real-time implementations of a full rotation feature tracking algorithm running on a low size, weight, and power (SWAP) platform fit for small UAV mounting.

1.3 Problem Space

Using an FPGA as a low SWAP means of accelerating homography estimation through SURF poses three key technical challenges. First, the previous implementations of SURF on an FPGA are limited to upright-only image comparisons, whereas the aerial domain demands feature detection with full rotation[24][31]. Second, previous approaches rely on external hardware or FPGA simulated processors to calculate the feature description which adds unnecessary overhead and limits parallel processing on the FPGA[5][24][31]. Third, previous approaches rely heavily on external memory and other components external to the FPGA, while a logic-only implementation may serve better to minimize the weight and power burden for small aerial platforms[5][24][31].

1.4 Proposed Solution

To overcome the technical challenges of low-SWAP aerial navigation, this research proposes Discrete SURF, a complete and robust SURF implementation for homography estimation that supports real-time aerial navigation tasks in the final meter of approach. Discrete SURF implements all of the stages of the SURF algorithm, from integral image calculation, to detection, orientation assignment, description, and matching. In answering the key challenges describe previously, the following technical contributions are claimed:

1. A robust implementation of SURF in FPGA logic that supports rotation during image comparisons for aerial movement.
2. A complete implementation of SURF detection, description and matching steps in FPGA that takes full advantage of hardware acceleration.
3. A logic-only implementation that allows for a small form factor as well as expedient FPGA to ASIC conversion.

The merits of Discrete SURF will be compared against the OpenCV SURF implementation for accuracy in detecting a landing platform represented by a quick response (QR) code. The detector stage of Discrete SURF will be compared to the OpenCV SURF detector to verify its accuracy in producing matches. Similarly, the combined Discrete SURF descriptor and matcher stages will be compared against their OpenCV counterparts to determine accuracy. A simulated scenario will be created to demonstrate the effectiveness of Discrete SURF for navigation within close distances of a target. Finally, an analysis of the algorithm's timing requirements at each stage and an snapshot of the overall Virtex-4 utilization are presented.

In Chapter 2, we analyze select computer vision approaches for close proximity aerial navigation tasks and review the state of the art of monocular passive vision and position estimation for aerial navigation. In Chapters 3 and 4, we describe the SURF implementation on a Virtex-4 FPGA and evaluated it as a viable platform for real-time (30fps) processing of images with resolutions of 320x240 pixels. In Chapter 5, the research is concluded and future work is laid out.

II. Background and Previous Research

In this chapter, the state of the art approaches in monocular passive vision and position estimation for aerial navigation are reviewed. The analysis flows from a general description of the difficulties in vision-based aerial navigation and moves toward how these problems are addressed by on-board computer systems. The range of potential solutions provided by the research focus of this study is bounded to a specific application for implementation on a small UAV. The scenario reveals a critical need for vision aided aerial navigation using feature tracking, as well as a processor for such that has low SWAP attributes. From there, the discussion focuses on properties of an FPGA as a platform to accelerate the tracking methods homography estimation required to determine relative position and orientation. The analysis concludes as current challenges in the area set the research task.

2.1 Scenario

One of the most difficult aspects of vision-aided aerial navigation is robust detection that can characterize targets as distances and orientations change during approach. A vision algorithm that can track a few pixels at long distance will generally not offer precision pose estimation at closer distances. For this reason, this research has characterized target detection into three distinct phases to be evaluated differently. The first phase, rendezvous, describes the initial flight towards a general area known to contain a target and the search for the target. The second phase, approach, consists of target tracking from 10 meters away to around 1 meter away. The final phase, close range navigation, is target tracking that occurs within one meter. The decrease in distance at each phase is accompanied by an increasing need for precise pose estimation to prevent a collision. The rendezvous and approach phases can make more generalized pose estimations but also have less target resolution to base their tracking on.

In addition to requiring precise pose estimation, aerial tracking also requires real-time processing to make quick position adjustments in the case of unanticipated movement cause by wind or other environmental factors. This requirement for real-time processing necessitates a capable on-board processor to compute position accurately and quickly enough to prevent collision and facilitate docking. On an airborne platform, larger processors diminish the mission capability due to their weight and power consumption. Therefore, pose estimation in the docking phase would have to be done by a powerful, on-board processor with low SWAP requirements. This study seeks to evaluate a plausible homography estimation implementation that meets the requirements of aerial navigation.

2.2 Analysis of Alternatives

A number of methods for close range navigation exist. We provide a brief overview to explain the rationale behind using homgraphy estimation based on feature tracking algorithms.

2.2.1 Inertial Data Communications.

Communication between platforms with their own on-board inertial navigation systems (INS) is a common solution to relative navigation that is cheap in both cost and SWAP requirements. Platforms with INS systems usually rely on their inertial sensors to augment their GPS coordinates to provide a more accurate estimation of movement and position. By communicating their enhanced pose estimation, many platforms can navigate relative to one another simultaneously[22].

The main problem with relying on inertial data communication for relative navigation is the reliance on trustworthy data in the system. Position estimation errors are amplified when transmitting pose belief between platforms without anyway to verify information. More importantly, inertial data communication cannot provide navigation with regard to scene objects that are not equipped to transmit information. To provide a more robust solution, we opt for vision techniques.

2.2.2 Stereo Vision.

Stereo vision relies on two cameras to provide different scene perspectives which can be used to calculate a depth estimation. When applied to relative navigation, stereo vision can estimate the distance and pose difference between the cameras and an object in the scene. The advantage of stereo vision is that it does not need any pre-existing knowledge of scene objects because it can calculate distance by comparing the distance between the same feature in both images to the distance between the cameras[2].

The main limitation of stereo vision as a relative navigation method is the increased size, weight and power (SWAP) requirements to run two cameras. Additionally, feature detection, description, and matching has a high computational requirement, which further increases the SWAP and overall costs of a stereo vision system over other solutions[2].

2.2.3 Optical Flow.

Optical flow is a monocular vision technique that uses the apparent motion of objects to determine their location. Optical flow can be measured using only a single camera and can provide velocity estimation between two or more scenes. Optical flow is found by computing the independent motion of every pixel in a frame to the subsequent frame[15].

The primary benefit of optical flow for relative navigation is that it is able to characterize the relative motion between the camera and scene objects without the need to transmit inertial data. Therefore, any movement detected by optical flow can be considered the sum of the relative movements when trying to bring a camera and an object together. Furthermore, optical flow is sensitive to motion on the scale of a single pixel, meaning that it can characterize scene movement at higher resolution or at greater distances than feature detection methods.

The primary problem faced by optical flow in relative navigation is the lack of scale information, and the inability to accurately characterize an objects orientation[15]. It is possible to evaluate the size of the blob representing a known object as detected by optical flow and then make a rough estimate of the relative distance to the object. Additionally, if the camera is able to be controlled, it is possible to orient to an optical flow tracked object by making movements that keep the object in the center of the image frame.

As we show in the next section, although optical flow provides a decent location estimation at longer distances, a more precise image processing method is required in order to accurately characterize the objects pose relative to the camera.

2.2.4 Homography Estimation from Feature Tracking.

Homography estimation from a single vision camera relies on tracked feature points from a predefined model to a scene image to determine the translation from the camera point of view the orientation of the model[7]. Homography is found by comparing features from an image with a known orientation to the same features of an image at a different orientation. The perspective lines that lead from the camera through the image plane and mapped to features in the scene can be compared to similar vectors for the model image in order to calculate homography, as shown in Figure 1.

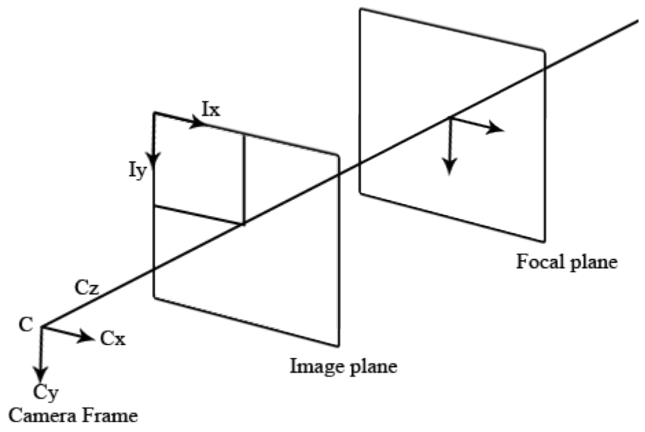


Figure 1. The z-axis of the camera frame runs directly through the image plane and into the focal plane. The transformation from the focal plane to the image plane can be made using homogenous vectors because the image plane and focal plane have the same orientation.

Optical flow can likewise be used across multiple frames to determine relative motion in the scene and provide robust navigation in changing environments, however it cannot provided the precision orientation estimation required for close quarters operations. Alternatively, homography estimation relies on a higher resolution model of the feature to be tracked and is therefore not suited for tracking at smaller scales (further distances). As a result, one possible means of navigating towards another visible airborne object is to rely on optical flow for the first part of the approach until the homography estimator is close enough to make a viable pose estimation.

Figures 2 and 3 show a simulated scenario in which a camera approaches a hovering target (QR code). Figure 2 shows a point in the approach towards the target at which optical flow has a clear lock on the target, but the homography estimator does not have consistent enough feature matching to make a pose estimation. In Figure 3, the approach is entering the final phase in which the homography estimator is able to give an estimated pose reading. At this point, the optical flow tracking is still clearly defined, but it provides a model that is much more difficult to draw accurate position

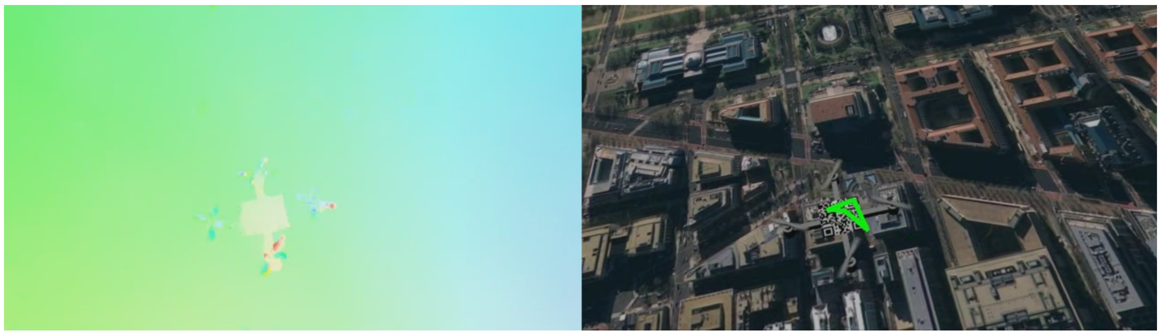


Figure 2. The current point of the approach scenario shows the homography estimation algorithm failing because of distance, but the optical flow algorithm has a solid look on the target.

estimation data from.

Since only a single camera is required for homography estimation, and computation is performed one frame at a time, it is a relatively low SWAP and computation solution. Although pose estimations that result from homography translations are usually less accurate than those generated by stereo vision, previous work has shown that monocular vision it is robust enough for navigation[24]. Overall, homography estimation from feature tracking provides both a cheap and robust approach to relative navigation and will be used in the experiments described in Chapter 3.

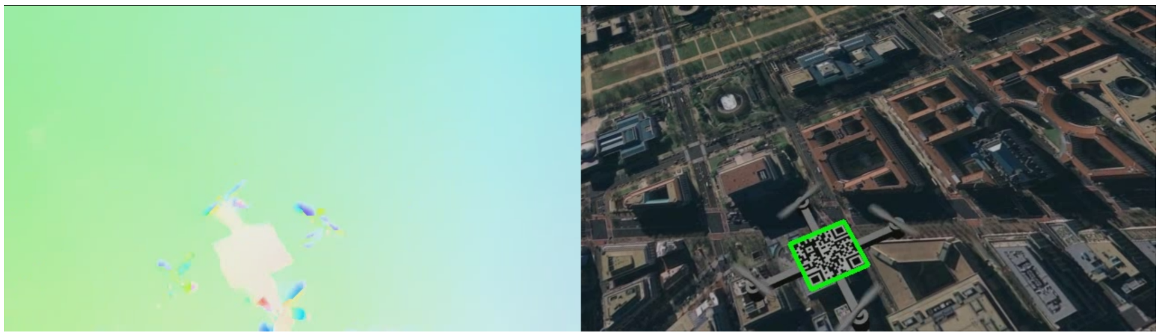


Figure 3. The current point in the approach scenario shows the optical flow algorithm still has a lock, but the homography estimator has enough information to give a pose estimate.

2.3 Feature Tracking Algorithms for Homography Estimation

A homography transform between the camera and an object being tracked can be used to calculate the relative distance, position, and orientation between the camera and object. The homography calculation requires that enough reference points from a known model are detectable in the scene image to determine the transformation between the model and the scene. In order to consistently find the model features as the model moves and rotates, it is necessary to have a robust method of tracking that can produce enough matches to the model with invariance to scale and rotation.

2.3.1 Feature Tracking.

Feature tracking in computer vision relies on features in an image that can be consistently matched from one scene to another. In order to effectively identify and describe these features in an image, a number of feature detection and description algorithms have emerged such as FAST[26], SIFT[14], and SURF[4]. All of these tracking algorithms work in three stages: feature detection, feature description, and feature matching.

2.3.1.1 General Method for Feature Tracking.

This subsection provides a brief overview of the phases of feature tracking. The first stage, feature detection, is where these algorithms differ the most. The scale-invariant feature transform (SIFT) algorithm is generally accepted as the most robust detection and description algorithm[16]. It relies on edge detection as a result of finding local minima and maxima of an image convolved by a difference of Gaussian functions[14]. The features from accelerated segment test (FAST) algorithm sacrifices robustness for speed and detects corners in an image by comparing the intensity of every pixel to a circle of 16 pixels around it in order to determine whether it is a corner. However, the FAST algorithm's reliance on pixel intensity makes it especially vulnerable to changes in scene lighting[21]. The speeded-up robust features (SURF) algorithm detects two-dimensional Hessian blobs in the integral of the original image-[4]. The Hessian blob is essentially a change of intensity from one level to a different level and then back to the original level. The result of calculating these local minima or maxima in the image provides the features coordinates and scale information needed for description.

Once a list of locations, orientations, and scales for each feature in the image is generated by the feature detector, the feature description stage begins by revisiting each of those locations to classify the feature in a way that that makes it robust against changes in the features geometry across scenes. The SIFT feature descriptor works by generating 160 different transform possibilities that consist of different scales at different pixel locations[14]. The high number of transform possibilities generated by the SIFT feature detector allows for greater accuracy, but significantly higher computational requirements. The FAST algorithm does not have a corresponding feature descriptor, but it is important to note that feature descriptors can be used interchangeably with a feature detector that outputs a compatible list of features. Accordingly, the FAST detection algorithm is often paired with the SURF feature descriptor. The SURF algorithm does have its own feature descriptor that is also designed with speed in mind. The SURF feature descriptor uses the Haar wavelet response based on the scale of the feature detected to determine feature orientation. The Haar wavelet is essentially the scale-normalized determinant of the same Hessian used in the feature detection stage of SURF. The result of SURF feature description is 64 descriptor unit-vectors that represent the Haar wavelet intensity of for each subregion of a feature[4].

The final stage in feature based tracking is feature matching. There are no special requirements for a feature matcher except that it needs to be able to correctly compare feature descriptions that have been extracted by the same algorithm. Most feature tracking algorithms rely on a nearest neighbor search that compares features from one scene to the next to determine if they are similar enough to be considered a match as determined by a specified match threshold. Approximation of nearest neighbors algorithms are often used for feature tracking because they return a good guess at matches between scenes with significantly less computation. The method implemented in the original SURF algorithm relied on the Euclidean distance between point descriptions that counted a match using a threshold distance[4]. The original SIFT algorithm uses the Best-bin-first algorithm to match points, which relies on building a tree based on point distances and then finding similar points from the nodes[14]. Both methods reduce the feature matching computation requirement, although the description stage is clearly the bottleneck in the feature tracking process.

2.3.1.2 SURF.

Compared to the FAST and SIFT feature detectors, the combination of SURF detector and descriptor is a good compromise. In addition to being several times faster that the SIFT algorithm, SURF has also been shown to to be more robust when applied to certain images[4]. Additionally, SIFT starts its detection at the highest detection setting, and works towards lower scales, which means that the minimum runtime for the algorithm will always be much greater than SURF, which detects across a predefined number of scales starting with the fastest[14]. Compared to FAST detection, SURF detects around ten times the amount of features which takes considerably more time to perform. However, in terms of highly texture environments it has been shown that more features lead to better tracking[11]. Therefore, in terms of finding a balance between runtime and number of matches, SURF provides a compromise as well as it own feature detector and matcher.

As SURF is a promising solution for robust feature tracking, the following is an in-depth explanation of the SURF algorithm. The SURF feature detector finds the feature location and generates a circular representation of the features scale without orientation information. The first step in detecting features with SURF is to calculate the estimated Hessian matrix of a point $p = (x, y)$ in an image I at scale σ. The Hessian matrix gives a discretization of the Gaussian and is defined as

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{yx}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{1}$$

where $L_{xx}(x,\sigma)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2}g(\sigma)$ with regard to point p[4]. The result of the Hessian of the Gaussian second order derivatives is an approximation of the image from which features can be determined. The primary speed up of the SURF algorithm comes from the use of the Gaussian

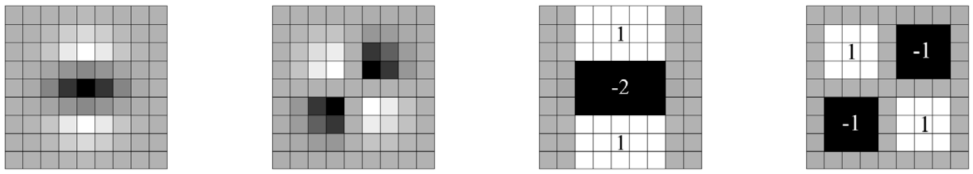


Figure 4. From left to right: the original Gaussian second order filters in the y-direction and the xy-direction, and the resulting approximation filter. The gray regions are discretized to zero.[4]

approximation filters and an integral image rather than an actual Gaussian filter. An integral image allows the sum of a block of pixels to be calculated with only four memory accesses rather than accessing every pixel in the block and applying a Gaussian value. The Hessian matrix for each scale is determined by applying an appropriate approximation filter scaled for size σ. In this way, features are found for different transforms of the original image and a variety of scales and orientations can be found to allow more precision in feature description.

The descriptor stage of the algorithm is the most crucial because features must have reproducible orientations and descriptors in order to produce matches. The description phase includes orientation assignment and descriptor calculation, both of which rely heavily on Haar wavelet response filters. The Haar-wavelet response is essentially a detector for transitions from black to white or vice-versa. The orientation is found by finding the Haar wavelet response in the x and y directions and choosing the angle around the point that corresponds to the largest summed response. The SURF descriptor then takes the area around a feature and divides it into 16 subregions that are oriented along the found orientation. Each subregion is assigned a four dimensional vector $v = (\Sigma d_x, \Sigma d_y, \Sigma|d_x|, \Sigma|d_y|)$ where d_x and d_y are the Haar wavelet responses in the x and y direction along the orientation angle. This results in a 64-dimension descriptor for each SURF feature, which can be tuned for invariance to contrast by turning the descriptor into a unit-vector.

Once the features are described, the matching is performed by calculating the Euclidean distance between descriptor vectors from different scenes. A pair is considered a match if the distance is less than 70 percent of the distance between the second nearest neighbor[4]. SURF-36 and SURF-128 are variants of SURF that use different lengths of descriptors to provide more (or less) expressive description vectors. Since SURF-36 uses a smaller description vector, features are described quicker but leads to a loss in matching precision. SIFT-128, on the other hand, matches slower, and takes longer to calculate, but Bay purports that it provides more accurate matches[4].

Another important variant of SURF is U-SURF, or upright SURF[4]. In U-SURF, a significant speed up is gained by avoiding orientation calculations. Additionally, because the description blocks and Haar wavelets are drawn along the pixel x and y-axes, the description can be done using the integral image rather than polling individual pixels. However, U-SURF comes at the cost of losing rotation invariance.

The baseline for performance and accuracy for the experiments in Chapter 3 is the OpenCV SURF implementation[6]. The OpenCV library is an open source library developed by Intel that allows access to each of the individual stages of the algorithm. Furthermore, the results from the FPGA implementation can be used in the OpenCV SURF algorithm by reading in data via serial port. By using the OpenCV library, this research will be able to verify the results of its implementation in terms of accuracy of detection, description and matching.

2.3.2 Model Fitting.

To address the problem of determining object pose in an image, model fitting techniques can search across a set of matched features to determine whether a model exists in an image, and if so how it is oriented. In order to get an accurate pose and scale estimation of a desired object in an image, it is necessary to have a model of a predetermined size that represents that object. The result of running the SURF algorithm on both an image representing the model, and the scene containing the object of interest is a set of feature matches between them. When a 2D model is found in an image, the transformation between the model and the scene object is known as the homography. The homography matrix gives the necessary transform to find the orientation of the object in the scene. One of a number of algorithms, such as a brute force approach or random sampling and consensus (RANSAC)[9] can be run on the set of matching features to determine the good matches and remove outliers before the homography is estimated.

The brute force approach for finding candidate features for homography estimation attempts to match all features in the set of matches. This approach not only leads to increased computation time, but it is also overly reliant on near perfect matching by the feature tracking algorithm. This makes it a poor choice for use with SURF features. The RANSAC approach offers a compromise in computation time while also letting the user define the amount of searching to perform. With RANSAC, a random selection of matches is chosen and the homography is computed. RANSAC will continue to run for a user defined number of iterations and will return the set of matches that resulted in the best homography, which is defined by the highest number of matches that were fit into the model. RANSAC is ideal for real-time solutions because it allows for a definable search time, and it is also able to make a match with only a few correct inliers[9].

The computation of a homography matrix H_{ba} follows from its definition:

$$H_{ba} = R - \frac{tn^T}{d} \tag{2}$$

where R is the rotation matrix from b to a, t is the translational vector and n and d are the normal vector and distance to the plane. The rotation matrix for a successfully fitted model is an estimation of the pose of the object in the scene while the translational vector and distance to the plane gives the relative position. The homography matrix is determined given a set of matches, p_i, between points in the scene image a and points in the model image b and the instrinsic camera parameter matrices for the scene image K_a and the model image K_b, using:

$$^{a}p_i = K_a \cdot H_{ba} \cdot K_a^{-1} \cdot^{b} p_i \tag{3}$$

The above algorithm is run iteratively with RANSAC. RANSAC determines p_i until the iteration limit set for RANSAC is reached, while homography estimates are produced at each step. The best homography estimate gives the estimated rotation and translation.

2.3.2.1 Quick Response Code as a Model.

The quick response (QR) code is a matrix barcode that was designed to be machine-readable. The template design consists of three distinct squares in the three corners to define the borders of the code, and a fourth smaller code in the remaining corner to normalize the scale. The smaller dots that make up the rest of the code are arranged in a manner that conveys the desired information to an informed reader.

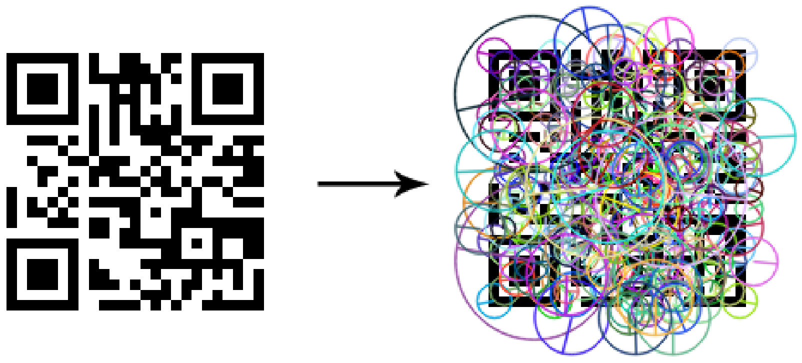


Figure 5. The left image shows an ordinary QR code while the right shows the number of SURF features with orientation and scale detected for the same QR code

Although the aforementioned feature tracking algorithms do not look at features as way to identify a QR code as a whole, the QR code lends itself well to feature tracking. Due to the QR code's black-on-white design, it is an excellent candidate to be tracked by the SURF algorithm in particular. The descriptor stage of the SURF algorithm uses the Haar wavelet response to describe feature orientation which consists of finding transitions from white to black or vice-versa. The result is a feature set that allows for large changes in rotation and translation between the QR code model image and a QR code in a scene. Furthermore, because of the pyramidal calculation of features by SURF, there are larger features detected in the QR code that allow the overall orientation of the code to be characterized. Overall, the traits of the QR code make it an excellent medium for determining the pose of an object it is mounted on.

2.4 FPGA

An FPGA is an excellent technology to accelerate homography estimation for aerial navigation. An FPGA has low SWAP requirements as well rich computational resources and flexible programmability. An FPGA was chosen because it gives a very high performance to size ratio and is therefore a viable option for a mounted solution on small UAVs. The additional hardware supplied on FPGA development boards such as RAM and peripheral I/O allows images to be accumulated for processing and makes it easier to interface external cameras and hardware with the processors. Furthermore, developing on an FPGA also provides a straight forward means of creating a specialized ASIC from the FPGA implementation.

2.4.1 Image Processing.

FPGAs excel at parallel processing, and are some of the best small-package solutions for image processing. Most image processing algorithms break an image down into pixels and perform simple, repetitive math for every pixel. The current state of the art image processing platform is a GPU, which relies on many math computational units to simultaneously analyze multiple pieces of a frame. Since the hardware is customizable on an FPGA, it offers a similar, although much less powerful solution than a GPU in that it is possible to create a hardware pipeline to parallelize many image processing algorithms including aspects of SURF and optical flow. Additionally, most modern FPGAs, including the Virtex-4, have anywhere from a dozen to a few thousand digital signal processing blocks, which are essentially math units that specialize in the simple math operations required for image processing algorithms. In addition to the DSP blocks, FPGAs also provide customizable logic blocks that can be used to implement other aspects of the image processing algorithms, but can also provide the overall architecture for carrying out the algorithms.

2.4.1.1 SURF on FPGA.

There have been a number of proposed FPGA SURF architectures as well as several that have actually been implemented. At the forefront of these implementations is the Svab design[24], FLEX-SURF, which is capable of running the entire algorithm from start to finish, although not entirely in FPGA hardware. This is the most complete implementation to date, however, there are many additional hardware requirements for the Svab design such as SSRAM, and a specialized camera board. Furthermore, this design relies on a Linux distribution as the platform to run the description calculation adding unnecessary overhead. Alternatively, the Bouris[5] implementation is done entirely in FPGA hardware, but only goes as far as the detection and orientation stages without producing descriptors and matches. Their implementation of SURF detection ran comparably fast to top of the line GPUs at 56 frames per second for standard video resolution (640x480 pixels). Another proposed FPGA implementation, HR-SURF, designed by Battezzati[3] is an FPGA architecture for the SURF algorithm from detection to matching, although the results given are theoretical estimates of performance, and they give no indication that their architecture was fully implemented on an FPGA. Similar to Flex-SURF, the Worcester Polytechnic Institute has shown an FPGA running the detection stage of SURF, while the chips PowerPC to ran the description phase[31]. The results of their system is 60fps feature matching at a resolution of 800x600 pixels.

To the knowledge of this research, the only FPGA implementations of the second stage of the SURF algorithm, description, have relied on processors simulated in the FPGA logic that allow for the floating point operations required by the original SURF algorithm. Furthermore, these implementations have not included description calculation with orientation assignment, which allows for rotation invariance greater than +/- 15 degrees. The previous work shows that FPGAs are more than capable of real-time feature tracking, although none of the implementations thus far have included the entire algorithm in the FPGA logic with the rotation aspect necessary for aerial navigation.

2.5 Conclusions

In conclusion, we summarize open challenges in SURF acceleration for homography estimators in aerial navigation. To date, there have not been any real-time implementations of a full rotation feature tracking algorithm running on a low size, weight, and power (SWAP) platform fit for small UAV mounting.

Using an FPGA as a low SWAP means of accelerating homography estimation through SURF poses three key technical challenges. First, the previous implementations of SURF on an FPGA are limited to upright-only image comparisons, whereas the aerial domain demands feature detection with full rotation. Second, previous approaches rely on external hardware or FPGA simulated processors to calculate the feature description which adds unnecessary overhead and limits parallel processing on the FPGA. Third, previous approaches rely heavily on external memory and other components external to the FPGA, while a logic-only implementation may serve better to minimize the weight and power burden for small aerial platforms. Therefore, the current state of the art is not taking full advantage of the capabilities of a FPGA SURF implementation with regard to aerial navigation. This review sets the research aims for Discrete SURF.

III. Implementation and Evaluation Method for Discrete SURF

This chapter describes the implementation of the SURF algorithm in an FPGA, and especially the special considerations and alterations that tailor the algorithm for an FPGAs unique requirements. First, an overview of the system architecture and algorithm flow is given. Second, each individual stage of the algorithm and how it is implemented is described. Finally, a method of evaluating the accuracy of SURF running on the FPGA will be explained.

3.1 System Overview

The original speeded-up robust features algorithm initially developed by Bay[4] requires special considerations for implementation on a field programmable gate array. Foremost of these considerations is the potential for parallelism offered by an FPGA, whereas the initial algorithm proposal is sequential. Previous FPGA implementations have relied on simulating processors in the fabric to follow the original algorithm at the cost of parallelism and greater overhead. The use of simulated processors allows for easier implementation as well as the use of simulated floating point constructs that are commonplace in most programming languages, but are awkward and wasteful when implemented in an FPGA. In many applications, it is also possible to completely ignore the rotation of features in an image and therefore only a small and inexpensive portion of the SURF description algorithm known as upright-SURF is implemented. However, in order to detect features with an aerial vehicle it is necessary to have the invariance to rotation provided by the full SURF algorithm which requires significantly more processing and can no longer rely on the benefits of the integral image. Furthermore, it is necessary to accurately match the SURF features, which has never been implemented on an FPGA.

3.1.1 Design Goals for SURF on FPGA.

The goal of this FPGA SURF implementation is to stay true to the principles of the original algorithm, while also providing parallelism and addressing the limitations of the FPGA in floating point operations. By implementing the entire algorithm in the FPGA fabric, there should be significant speed-up, especially in the descriptor stage, over implementations that rely on simulated processors. Rather than waste FPGA resources by simulating floating point operations in order to apply Gaussian filters in the description stage, this implementation will apply a Gaussian mask with quick bit shifts. Upright SURF will be a parameter that allows for quick description without rotation, but the full rotation description is implemented with predefined rotation masks for all possible feature scales at increased speed costs but better matching. This specialized SURF implementation combined with the homography estimation results in a system that is able to reliable estimate position in real-time.

3.1.2 Hardware Environment.

The board being used for the experiments is a Xilinx ML402 development board which features the Xilinx Virtex-4 SX35 FPGA first manufactured in 2005. The SX version of this chip features a higher number of DSP blocks than other versions in the Virtex-4 family, but also has less block RAM and no PowerPC on board. At the time of writing, the current version of the Virtex chip set, the Virtex-7, has roughly 10 times the RAM, 20 times the logic cells, as well as improved clocking performance. In order to maximize the portability of this implementation for use on other FPGAs, the design has been created completely in logic with the exception of the use of GPIO pins broken out on the ML402 board which interface to a OV7670 camera module.

FPGA internal BRAM is more expensive per bit than external memory and therefore it is recommended that external memory be considered forh Discrete SURF before upgrading to a more powerful FPGA.

3.2 Discrete SURF

The following implementation of the SURF algorithm on an FPGA has been named Discrete SURF after the special considerations made to accommodate the FPGAs unique requirements. The main difference between the original SURF algorithm and SURF is the use of masks to approximate pixel calculations that would otherwise require complex floating point operations which are non-native to an FPGA. Other design changes were made to facilitate parallel processing and fully utilize the FPGAs resources.

3.2.1 Algorithm Overview.

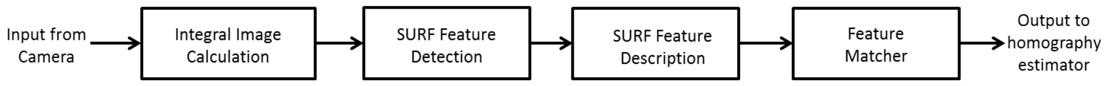


Figure 6. The four stages of Discrete SURF

Discrete SURF accepts a camera frame or preloaded image as input from RAM and outputs coordinate pairs and distance weights for the best matches. An overview of the stages of SURF can be seen in Figure 6. From left to right, the image data is first sent to the integral image module to produce a summed area table of pixel values. Once the integral image stage is finished, the detection stage uses the integral image values to determine feature coordinates and scale. After the detector has finished, the descriptor stage can begin and assign an orientation and descriptor vector to each feature based on the integral and captured image data. Finally, after the descriptor has described all of the features, the matcher phase compares the descriptors between images and outputs a matches list.

Figure 7 gives a deeper overview of the interaction between algorithm components. The user decides how the image is fed into the system, but the Discrete SURF module is interfaced to a single port RAM read in its current iteration. The user can also set parameters for upright SURF, number of detection levels and octaves (detection scale), image resolution, and Hessian determinant threshold. If upright SURF is being used, the image can be replaced as soon as the integral image is calculated because the description phase will be able to calculate response in blocks rather than needing individual pixel values. The integral image stage begins once the image capture has finished being saved to memory. The integral imaged reads an 8-bit vector from the image capture RAM, performs the integral image calculation, and writes the value to the integral image RAM. Once the integral image stage is complete, the detector stage begins by calculating the Hessian matrices for every pixel in the image based on the values stored in integral RAM. After the detector stage has determined which pixels are relevant features, the feature x and y coordinates as well as the scale and Hessian sign information are stored to the detection RAM. If the upright SURF flag is set true, then the descriptor stage skips the orientation assignment and determines descriptor values based in the integral image values. Otherwise, if upright SURF is not set, the descriptor begins by assigning orientation vectors to the features at the coordinates saved in detection RAM based on the integral image values and then finds the descriptor values based on the captured image RAM pixel values. The final stage, the matcher, takes the descriptors, as well as the feature coordinates that they correspond to, and matches them to the descriptors calculated from the target image. If the target image is known before hand, it is also possible to pre-load a list of descriptor values so that the entire target image and its integral image do not need to be saved to memory. The best match for each feature can then be saved to match

RAM, or output to the user.

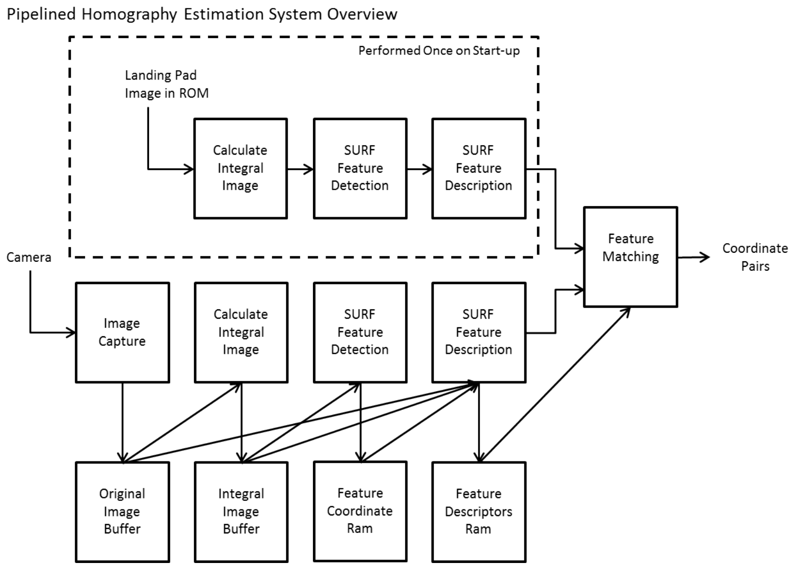


Figure 7. From left to right, the image is captured and then saved to RAM. The integral image is calculated from the captured image RAM and stored in its own RAM location. The detector stage uses the integral image to calculate the feature coordinates, which are saved in their own RAM. The descriptor stage uses the captured image RAM, integral RAM and feature coordinate RAM to calculate a list of descriptors which are saved in their own RAM. The matcher uses the descriptor RAM and compares it to the descriptor RAM of the landing pad image to determine matching feature coordinates which are output to the user.

Each of the stages of the SURF algorithm and its implementation on the Virtex-4 will now be explained in detail.

3.2.2 Integral Image.

Image processing algorithms often involve running the same operation on many if not all of the pixels in an image, for this reason, it is generally a good idea to pre-process any aspect of the image to limit the number of pixels processed or the time to process each pixel despite adding some upfront processing time. The integral image allows the SURF algorithm to determine the response about a pixel at any scale in only four memory accesses and with only one calculation rather than polling every pixel in the block for its value. SURF can use the integral image at up to three stages and therefore it is a necessary aspect of the algorithm for quick runtime. The

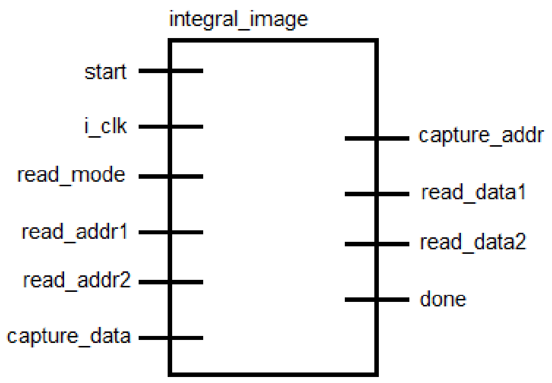


Figure 8. The integral image stage accepts the captured image data from RAM, and outputs the calculated integral image values.

integral image is a summed area table of pixels in the original image, meaning that the integral image of any value is equal to the sum of all pixels above and to the left of it in the original image as shown in Equation 4. Once the integral image is calculated, the sum of a block of pixels is the value of the bottom right pixel and the top left pixel minus the top right and bottom left pixels as shown in Figure 9.

$$I(x,y) = \sum_{x' \,\leq\, x, y' \,\leq\, y} i(x',y') \tag{4}$$

In the FPGA implementation, the pixel values are read into the integral image stage sequentially with the pixel column and the start of each new row tracked. The integral image is calculated at max speed by reading the pixels in sequentially, rather than create a 2D variable to store the whole image. The integral image value that is saved to the integral image RAM is equal to the current pixel value incoming from the captured image ram, plus the sum of all the pixels in the current row of the image and the integral image value that was saved at the same column for the row above the current row. In order to process the pixels sequentially, it was necessary to create a structure in logic that holds the previous rows integral image values that is updated as new values are calculated.

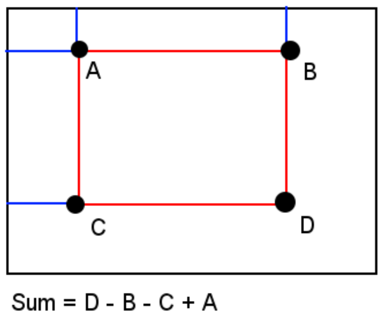


Figure 9. The integral image allows for the sum of a group of pixels to be found with only four memory accesses, and a single calculation.

Data access from the integral image is a potential bottleneck for the SURF algorithm because four integral value accesses are required per response calculation. In order to mitigate slowdown, the integral image RAM is implemented as a dual port RAM. During the integral image calculation stage, the integral image RAM is in write mode and uses one of its ports to record the integral calculation data. After the integral image calculation stage, both of the integral image RAM ports are set to read mode, which allows for simultaneous reads for two out of the four reads required to calculate the sum of a block of pixels. The Virtex-4 allows for up to two read/write ports per RAM, but newer FPGAs allow for quad-port RAM which would enable all four access simultaneous. Alternatively, an FPGA with more block RAMs than the Virtex-4 could divide the integral image into multiple RAMS for more simultaneous reads.

3.2.3 Discrete SURF Detector.

The Discrete SURF detector stage received the integral image data as an input, and outputs a feature vector containing coordinates, scale, and sign as shown in figure 10. The primary purpose of the detector stage is to find to best feature coordinates and scales that the descriptors will then be centered around. The SURF detector is a blob detector rather than a corner detector, and generates the largest response for

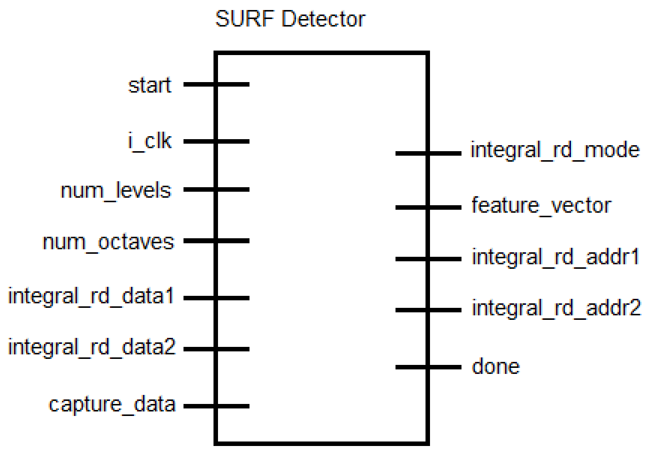


Figure 10. The SURF detector module takes in integral image data and detects feature points at the scales specified by the num levels and num octaves parameters. The output from the detector is a 25-bit feature vector that contains the feature x, y, scale, and sign.

points in an image that experience changes from high to low to high intensity in both the x and y direction. In some blob detectors, this response is calculated based on a series of Gaussian filters centered around the pixel of interest. The SURF algorithm receives much of its speed up over other detectors by discretization of the Gaussian filters and then using the integral image to quickly calculate them. The result is that a pixel can be evaluated at any scale in the same amount of time, despite larger filters being applied at larger scales.

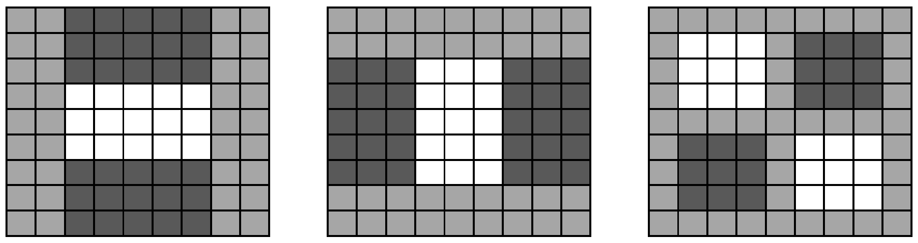


Figure 11. These represent the lowest scale filters, 9x9, that are used to determine Hessian response. From left to right, the filters determine D_y, D_x, and D_{xy} responses around the pixel of interest. Grey blocks represent a zero mask, white applies a positive mask, and black applies a negative mask.

Discrete SURF starts by analyzing the first pixel in the image that has a width and height greater than half the smallest filter size. Any pixels before this point will be too close to the edges of the image to properly calculate the Hessian response. For each pixel, three filters are applied, one each for D_y, D_x, and D_{xy} as shown in Figure 11. Three integral image summations are required to calculate the total response for D_y and D_x responses, and four summations for D_{xy} since there is an additional box. Once the overall response is found, it is necessary to divide the overall response by the number of pixels in the filter to get the average response. The divides for the response are done in parallel. The average responses for D_y, D_x, and D_{xy} are used to calculated the estimated Hessian determinant as shown in equation (5).

$$det(H) = D_x * D_y - (0.875 D_{xy})^2 \tag{5}$$

The D_{xy} response is ordinarily normalized to the D_y and D_x values by multiplying by 0.9[4]. This implementation seeks to avoid using floating point operations and unnecessary divides and as a result, uses a modifier of 0.875 which is equivalent to multiplying by seven and then bit shifting three to the right.

Although the response calculations are bottlenecked by the integral image memory accesses, the detector module has been implemented so that the second stage of detection is being performed in parallel. Non-maximal suppression is performed on all of the Hessian determinants in the image to find the local maxima. If the magnitude of the local maxima is above the Hessian threshold set by the user, then the point is considered a detected feature. Non-maximal suppression takes a 3x3 block of pixels centered around the current pixel of interest for all levels in the current octave as shown in Figure 12. If the Hessian determinant of the pixel of interest has a greater magnitude than any of the pixels surrounding it in its level and all of the pixels

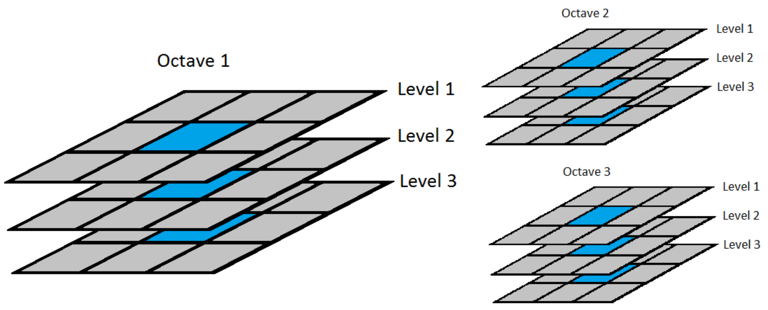


Figure 12. The non-maximal suppression step of the SURF detector finds local maxima for the levels within an octave. A 3x3 block of pixels are retrieved from memory and their Hessian determinant values are compared. If the center pixel of the 3x3 block is greater than the other eight in its own level, as well as greater than all other pixels in the 3x3 blocks of the other levels in the octave, then it is considered a feature. It is possible for there to be multiple features at the same image coordinates if they are in different octaves.

in the 3x3 blocks around other levels in the octave, then it is a feature. NMS is performed for each octave, so it is possible for there to be two features at the same x and y coordinate, however they will have very different scales. In order the perform NMS in parallel with the response calculation, a RAM module was designed to hold the previous three image rows worth of Hessian determinant values. Every time the response is calculated for an incoming pixel, the pixel in the same image column and two rows before is being evaluated by NMS as shown in Figure 13.

When a feature is found, the x and y image coordinates, scale and the sign of the Hessian Determinant are saved to feature RAM for use by the descriptor and matcher. However, for Discrete SURF, there are a few special considerations and changes to the feature values before it is recorded. The original SURF implementation defines the scale, σ, as a floating point number dependent on the filter size. In order keep this implementation discrete, the scale for each filter size is rounded to the nearest integer before it is saved alongside the feature. Additionally, because of the special properties of the SURF descriptor and the masks it uses, the feature is only recorded if the x and y coordinates are each greater than fourteen times the discretized scale

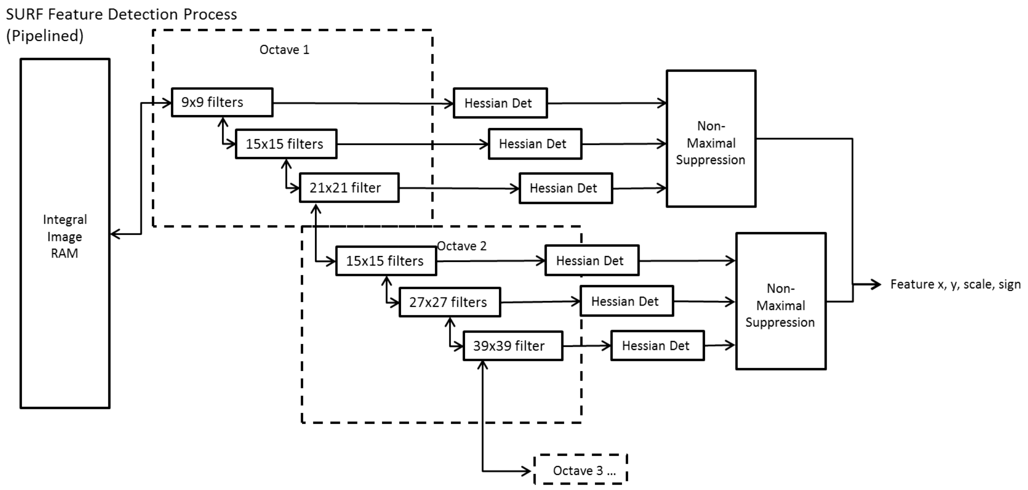


Figure 13. The SURF detector pipelines the response calculations for the different scales so that it is possible to simultaneously perform non-maximal suppression on previous Hessian determinant values while calculating the Hessian response values for the next pixels.

in order to fit feature description within the bounds of the image.

3.2.4 Discrete SURF Descriptor.

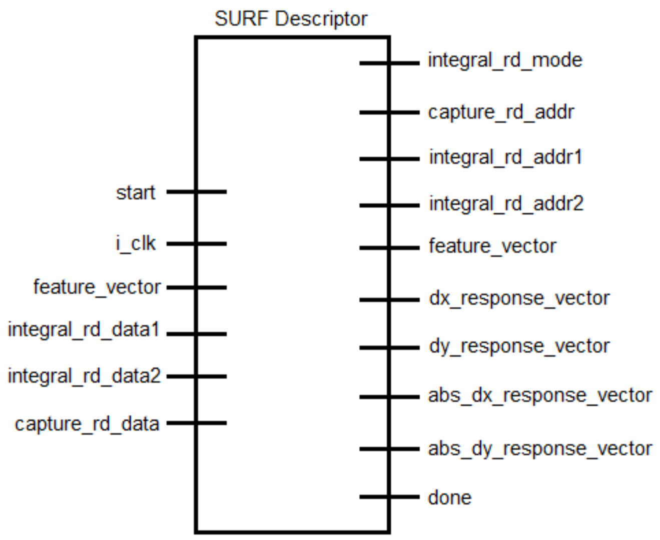


Figure 14. The SURF descriptor module reads information from both the integral and raw image RAM. The first stage of the descriptor calculates the orientation of the detected feature, and the second stage determines a 64-dimension descriptor vector that describes the response around the feature at the found orientation. The descriptors are written to RAM for use by the matcher.

The description stage is the most important stage of SURF because incorrectly characterizing the orientation of a descriptor will always lead to improper matches, regardless of how good detected features are. The descriptor stage is divided into two steps, orientation and descriptor calculation. If the upright SURF flag is set by the user, then the orientation step is skipped and the descriptors can be calculated much more quickly at the cost of losing rotation invariance.

In the original SURF algorithm, feature orientation is found by analyzing the pixels in a circle with a radius of 6σ around the feature. The D_x and D_y Haar wavelet response is found for a wavelet with size 4σ a side is found for each pixel in the circle with a sample size of σ. The sum of the two responses in either direction at each pixel are plotted with D_y in the y-direction and D_x in the x-direction as shown in Figure 15. Once all of the response magnitudes are calculated, a Gaussian is applied to give the response nearer to the feature a higher weight. The responses are binned into bins of 60 degrees based on their angle around the plot and the largest bin is chosen as the angle for the feature.

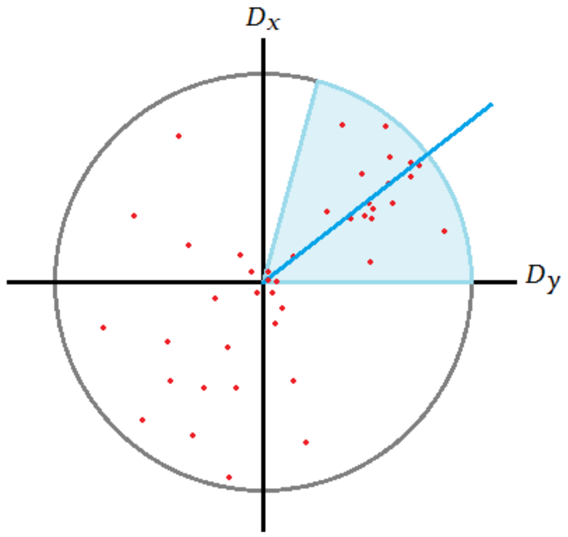


Figure 15. The orientation step of the descriptor stage uses integral image information to calculate the D_x and D_y response for pixels in a 6σ radius around the feature. The magnitudes of the responses for each pixel in the circle are plotted based on their D_x and D_y values and the plot is polled in 75 degree sweeps to determine the largest response indicating the orientation.

The same principles are followed in Discrete SURF in that a circle of pixels is found around the feature with a radius of 6σ, and because the scale was rounded to the nearest integer before being recorded in the detector stage, the circle can be represented as a predetermined mask shown in Figure 16. The number of pixels in the circle is the same regardless of the scale, but the sample size (scale) determines the number of pixels between each pixel that will be evaluated for the orientation calculation. The Haar wavelet is applied in the x and y and the magnitudes of the resultant response vectors are summed. Rather than apply a complicated Gaussian filter requiring floating point calculations to the summed response vectors, the mask in Figure 16 is used to approximate the Gaussian's effect. Each color on the masks

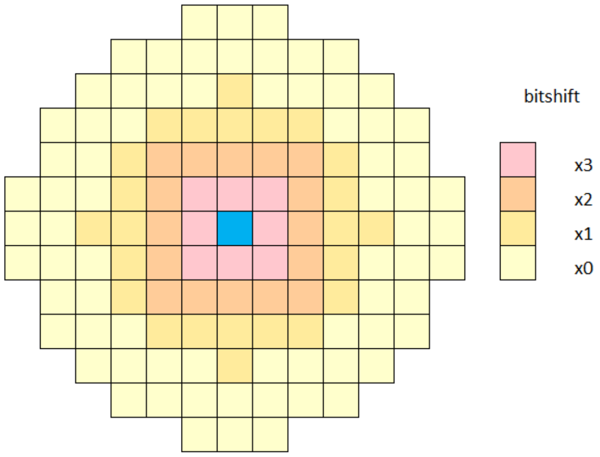


Figure 16. The orientation magnitudes are weighted with an estimated Gaussian mask that gives a higher weight to the values towards the center of the feature in order to find a more reproducible orientation that is not subject to outliers.

corresponds to a different number of left bit shifts that drastically increase the magnitude of the response nearer to center. The overall magnitude for the response of each pixel is then binned into one of 24 angle bins that correspond to 15 degrees increments around the circle. In order to calculate the angle of each pixel response without the use of floating point numbers, the response quadrant is first determined based on the sign of the D_x and D_y responses. Then, the D_y response is multiplied by eight before being divided by the D_x response to find the slope. By multiplying the D_y response before dividing, the resultant slope integer is also multiplied by eight and can be used to bin the response into one of six bins within the quadrant bin. If the D_y response had not been multiplied, the resultant slope for any angle less than 45 degrees would have been a floating point result and been unusable.

Each of four quadrant bins, plus an additional six angle bins per quadrant creates 24 overall angle bins. Although the original SURF algorithm indicates that they found the best sweep angle to be 60 degrees through experimentation, this implementation has found the best sweep angle to be 75 degrees through its experimentation. Five adjacent bins are evaluated at a time to sweep across all of the response magnitudes at 75 degrees incremented by 15 degrees. The angle of the center bin of the five that create the largest response is used as the feature angle in the descriptor calculation.

Upright SURF has the speed up benefits of being able to completely skip the orientation stage, and also to use the integral image method of quickly calculating the sum of blocks of images. Next, for U-SURF or SURF, the SURF description step analyzes a 4 by 4 block of smaller 5σ by 5σ blocks of pixels centered on the feature point. Each of the 4 by 4 blocks is assigned a section number one through sixteen and four response values are calculated for each section creating a total of 64 response vectors. Since the D_x and D_y Haar wavelet responses are calculated at every pixel in every section, the integral image is used extensively when the feature angle allows for the sections to be in vertical and horizontal blocks. However, at any angle other than an increment of 90 degrees, the 4 by 4 section block, as well as the axis for calculating D_x and D_y Haar wavelet response are now oriented along the feature angle, making it impossible to request sums of blocks of pixels from the integral image. For angled features, individual pixels requested from the original image are requested in determining the response. Furthermore, the math required to determine which pixels are required to calculate D_x and D_y responses at different scales is extensive and requires many floating point operations.

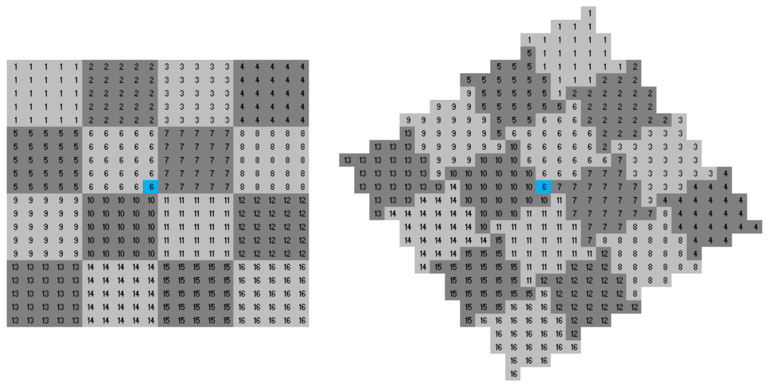


Figure 17. A predefined description mask is used to define the pixel groups the represent the 16 descriptor sections. On the left is the description mask used for upright SURF. On the right is the description mask used for feature angles 45, 135, 225, and 315. The same mask can be used for increments of 90 degrees by applying the mask from the four different corners.

In Discrete SURF, the description vectors for rotated features are calculated based on predefined masks for the 16 descriptor sections and the Haar wavelet masks at the different scales. By using masks, there is no need to calculate which pixels belong to which section as well as what pixels belong to which Haar wavelet at which scale and angle. Furthermore, because those calculations produce floating point results that must then be approximated into discrete pixel values, it is difficult to be sure that the Haar wavelets and the descriptor sections will always consist of the correct pixels, and the correct number of pixels. The masks ensure that for every orientation, the response is predictable and repeatable. A total of five descriptor sections masks as seen in Figure 17 are loaded onto the FPGA. The masks are size 28 by 28 and correspond to angles of 0, 15, 30, 45, 60, and 75 degrees.

Depending on the quadrant of the feature angle, the masks are applied reading from a starting corner to the opposite. For example, in the negative y, positive x

quadrant, the masks are applied from bottom left to top right to give angles from 90 to 180 degrees. The masks do not need to be scaled for size, rather the pixels in the image that corresponds to pixels in the mask are sampled at σ. The Haar

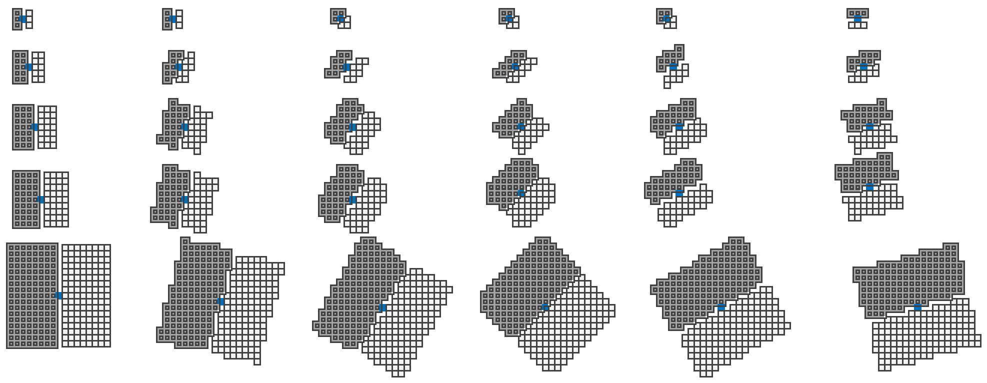


Figure 18. The Haar wavelet masks are predefined for all possible scales and rotations at 15 degree increments. The far left column shows the Haar wavelets used in upright SURF which can be calculated much more quickly using integral images. For features with orientation, each pixel in the Haar wavelet mask must be accessed to calculate the response, leading to dramatically increased processing time for larger scale features.

wavelet masks follow the same principle of reading from a certain corner to the other depending on the feature angle quadrant, however, special masks did have to be made for every possible scale. Since it is difficult to accurately represent a certain angle when limited to discrete pixel resolutions, five scale masks for each of five angles were made that represent the best possible representation of the angle at a scale as shown in Figure 18. At larger scales, the time to process Haar wavelet response for angled features increases greatly because a much larger mask is applied at every one of the pixels in the section mask. The total Haar wavelet response must be divided by the number of pixels in the wavelet in order to get an average response, because the number of pixels in a wavelet remains the same for an entire feature. This operation is done only once on the final response vectors of the section. Each wavelet response is also subject to a Gaussian centered around the feature so that responses near the center have a greater impact on the feature descriptor. This Gaussian has also been

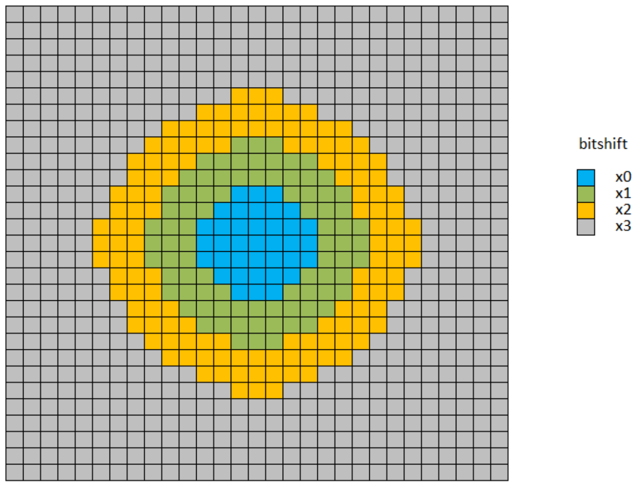


Figure 19. The Haar wavelet responses found at each pixel in the description mask are subject to a Gaussian to give a higher weight to responses nearer the center of the feature. This Gaussian is discretized into the above mask which uses right bit shifts (indicated by color) to scale the responses in accordance with its closeness to the feature center.

approximated as shown in Figure 19 where the color represents the number of bit shifts to the right that are applied to each response before it is added to the overall response vector for a section.

The total D_x and D_y responses as well as the absolute values of the D_x and D_y responses for an entire section are each recorded in their own vector giving sixteen sections worth of D_x, D_y, $|D_x|$, and $|D_y|$. The absolute magnitudes are included in the description to add invariance to changes in the polarity of the intensity. Invariance to scale is achieved by converting the response vectors to unit vectors, which is done when the vectors are divided by the number of pixels in the Haar wavelet filter.

3.2.5 Matcher.

The matcher stage is not an official component of the SURF algorithm, but it must be capable of accepting SURF descriptors from two images along with the feature coordinates and then output the feature coordinates for matches, so it is inherently linked. The original SURF paper uses the nearest neighbor ratio strategy to compare

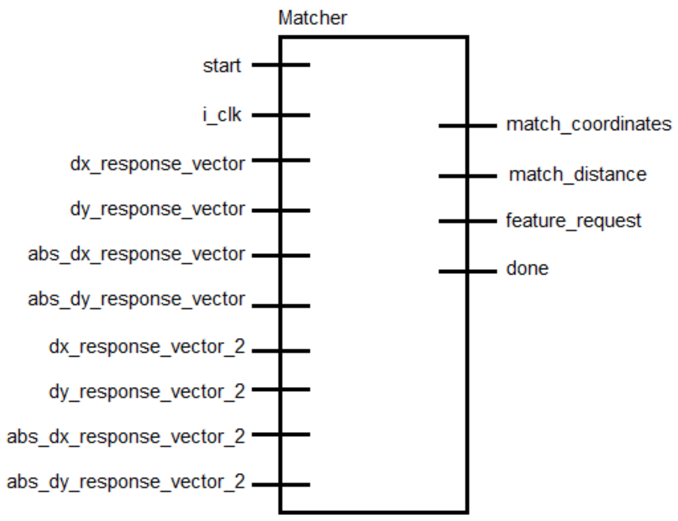


Figure 20. The feature matcher receives feature descriptors, as well as feature coordinates and sign from the features found in the new image, and compares them to the descriptors of the target image. The resulting best matches for each feature, as well as the distance, are output to the user for use in calculating homography.

descriptors, which consists of calculating the Euclidean distance between a pair of features and then considering a pair a match if their distance is 70 percent closer than the distance to next closest feature. The end goal of Discrete SURF is to provide the information need to make an accurate estimation of the homography transform from one image to another. For this reason, the matching was done differently than it was by original SURF.

The Discrete SURF matcher compares the first feature from one image to all other features in the second image and repeats this process until all features have been compared between the images. Since it does not matter what the actual distance between features is, but only that the magnitudes are represented proportionally, the Manhattan distance is used rather than the Euclidean distance. This is quicker to calculate and therefore improves the performance of the matcher. Four separate RAMs are used to store each of the four descriptor vectors for the features so that D_x, D_y, $|D_x|$, and $|D_y|$ can be accessed simultaneously for one section of a feature, improving parallelism. Once the total distance between the vectors of all sixteen sections is found for the pair of images, the distance and feature pair coordinates are only recorded if the distance is the shortest so far for the features in the pair. The result of the matcher is a pair of coordinates and a distance for each feature in the image that is been compared to the target image. The determination of good matches versus bad matches is left up to the model fitting stage of the homography estimator.

3.3 Implementation Validation and Analysis

The goals of Discrete SURF are to tailor the original SURF algorithm to FPGA fabric in a way that provides algorithm speed up, but also to maintain the properties and performance of the original algorithm. Therefore, Discrete SURF will be compared to the algorithm running on CPU and evaluated in terms of the performance of detected features, the accuracy of the descriptor and matcher, and the overall run time. Furthermore, this complete SURF implementation will be compared to the results of the previous work with partial FPGA implementations.

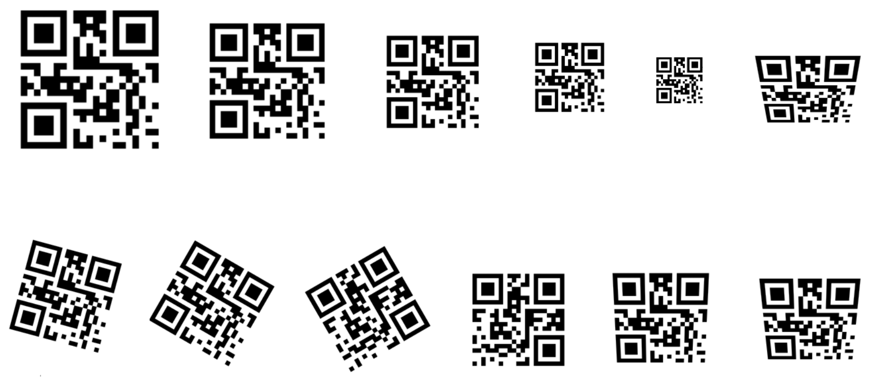


Figure 21. This image set was created to validate the properties of the SURF algorithm, namely scale and rotation invariance. A QR code is used as the feature set in the images and is rotated, scaled, and skewed to simulated multiple view angles for detection.

The data set that was chosen for this experiment consists of eleven 320x240 images that represent a QR code at different scales, rotations, and affine transforms. The data set was created with the intention of finding the limits of detection as well as verifying the correctness of the FPGA implementation. The images for rotation test at 15 degrees and then at 30 degree angle increments from 30 to 90 degrees. The affine transform image set tests for detection s the image is rotated from the top of the image to the bottom image in 10 degree increments from 10 to 30 degrees. The scaled image set test for scale invariance from 150% image size to 50% image size in 25% increments, excluding 100% scaled.

3.3.1 Detection Verification.

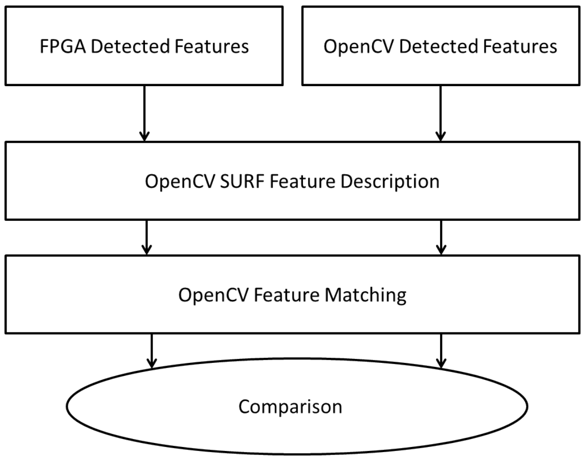


Figure 22. In experiment 1, the FPGA generated feature detection coordinates and scale are passed through the OpenCV SURF implementation and compared to the results from using OpenCV generated points.

The first set of experiments, depicted in Figure 23, evaluate the correctness of the individual aspects of the SURF implementation as well as the whole process. The correctness of the results will be compared to the OpenCV CPU SURF implementation to verify their correctness. The first stage of SURF to analyze is the detector phase, and although it is not expected that the detected points will be exactly the same as those found by OpenCV SURF, the detected points should be chosen so that they provide useful information to the descriptor phase. In order to verify that the detector produces useful points, the detected points from the FPGA implementation are plugged into the OpenCV descriptor and matching algorithm to test if they produce a similar number of correct matches to the OpenCV detected points. An image is first loaded onto the FPGA from MATLAB via a serial connection, then the detector stage is run on the image in the FPGA. Once the FPGA indicates that the detector stage has finished, the points are sent over the serial connection to the PC where they are plugged into the OpenCV description and matching algorithm. OpenCV shows the matches found using the FPGA and OpenCV detectors and the number of correct matches is visually counted for each. A similar number of correct matches from each algorithm indicates a functioning FPGA detector.

In order to ensure a fair comparison, the OpenCV and FPGA implementations will be configured to evaluate the image at three levels for a single octave. For each of the images in the eleven image data set, the number of correct matches produced by the OpenCV matcher will determine the effectiveness of the detector.

3.3.2 Description and Matching Verification.

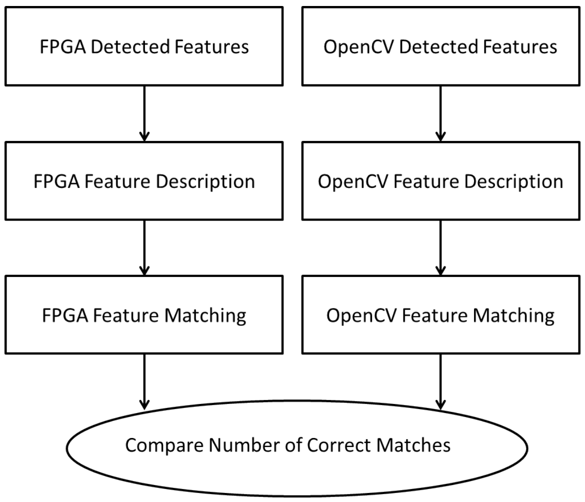


Figure 23. In experiment 2, the FPGA generated feature matches are compared directly to OpenCV generated feature matches to verify the accuracy of the FPGA implementation.

The second set of experiments evaluate the ability of the descriptor and matcher to match features between images with different view angles and distances. Once the effectiveness of the detector is verified, the FPGA descriptor and matcher can be tested for feature matching capability against the OpenCV SURF implementation. The FPGA receives an image from the image set via serial cable, runs the detector, descriptor and matcher and then outputs the list of matches to PC. The matches produced by both OpenCV and the FPGA implementation are then compared based on the number of correct matches. The parameters for the OpenCV and FPGA implementations will be configured to evaluate the image as similarly as possible. The runs for both implementations will use a single octave of three levels. The Hessian determinants will be calibrated before hand on the base QR image so that they produce a similar number of features before the experiment runs. For each of the images in the eleven image data set, the number of correct matches produced either method will be the measure of its effectiveness.

3.3.3 Timing Analysis.

The third set of experiments evaluate the speed performance at each stage of the algorithm. The speed of the algorithm is dictated by the scale dictated by the user and the resulting number of features found. The Discrete SURF module is configured to accept the number of octaves and levels as a user parameter. In the current implementation, the user can select up to three octaves with three levels each for a total of nine detection sizes with increasing processing time. Although the number of features detected for every scale will vary in a normal run of Discrete SURF, for this experiment the number of features per image is set to 50 regardless of scale to give an accurate comparison.This experiment will evaluate the total worst case and best case times for each size, as well as the individual times for the stages. Furthermore, the estimated worst case and best case frames per second will be calculated based on the total algorithm times. The goal of this experiment is to show the real-time capabilities of Discrete SURF on an FPGA. Our target frame rate is 30 fps, which is commonly regarded as real-time image processing, although other FPGA based SURF algorithms have claimed 10 fps as real-time[24].

3.3.4 Scenario Testing.

The final set of experiments analyze Discrete SURF in terms of its contributions to the domain of aerial operations. In order to make this evaluation, a simulated tracking scenario is presented. As shown in Figure 24, a UAV mounted camera is approaching a quad-rotor mounted landing platform. The platform is modified to display a QR code, although any target image with robust features could be used. As the UAV approaches the target image and enters the detection region, the calculated homography relative to the target image is used to direct the final meter of approach. In order to make this approach safely, the detection must be robust enough to be consistent as well as accurate.

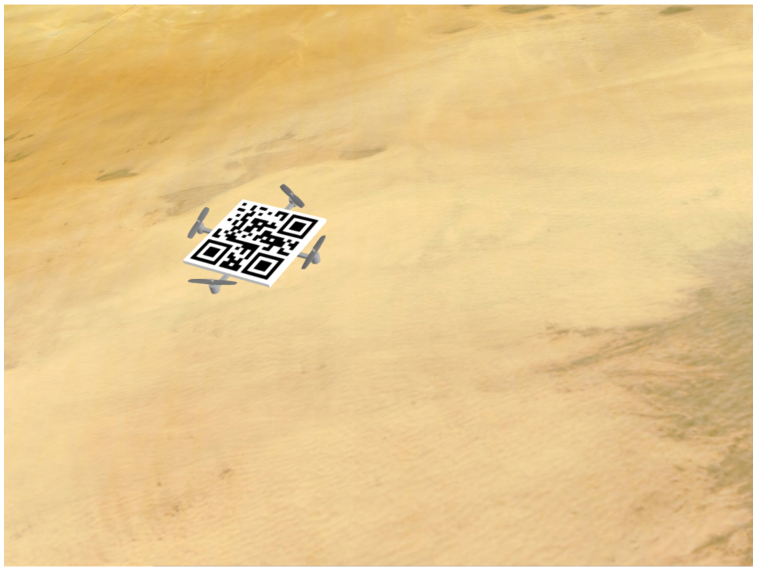


Figure 24. This scenario image shows the QR code painted on the airborne landing platform that will be approached by a UAV with a mounted camera. In the final meter of approach towards the platform, the homography estimation resulting from Discrete SURF matches will be determined.

The OpenCV toolbox will be used to show the homography transformation detected from the feature set. A good detection is shown by a box drawn around the QR code image that directly maps to the corners as shown in figure blah. The effectiveness of Discrete SURF for homography estimation will be determined by sending various scene images taken at different angles to the FPGA via serial cable and then running the OpenCV homography estimator on the calculated image matches. A successful detection is determined based on the accuracy of the drawn homography box.

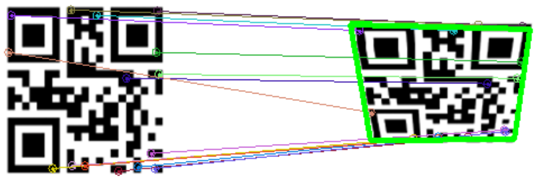


Figure 25. OpenCV takes a set of feature key point matches and uses the RANSAC algorithm to determine a homography transform, which is then presented visually as a box around the model in the scene image.

3.4 Conclusions

This implementation describes Discrete SURF, the first known implementation of the entire SURF algorithm with rotation in FPGA logic. The alterations to the original SURF algorithm designed by Bay were made in order to tailor the algorithm for an FPGA. As a result, the Discrete SURF makes use of pipelining and parallel processing of the images, but has discretized the process in order to avoid floating point operations which are not native to FPGAs. The experiments are designed to show that although modifications have been made to the original SURF algorithm, Discrete SURF still maintains the accuracy of the original algorithm and also contributes to performance and SWAP. Finally, we assert Discrete SURF's value in aerial navigation by presenting how the algorithm would operate in flight.

IV. Results and Analysis

This chapter presents and analyzes the results and accuracy of Discrete SURF on the FPGA. First, the detector stage of Discrete SURF is compared to the OpenCV SURF detector to verify its accuracy in producing matches. Similarly, the combined Discrete SURF descriptor and matcher stages are compared against their OpenCV counterparts to determine accuracy. Next, we show an analysis of the algorithm's timing requirements at each stage and a snapshot of the overall Virtex-4 utilization. Finally, a proof of concept for an in-flight scenario is presented.

4.1 Detection Verification Results

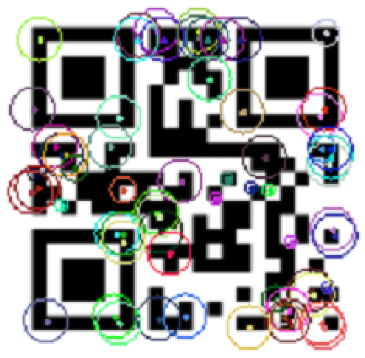


Figure 26. Features detected by the FPGA detector are sent back to the PC through a serial connection and plotted in MATLAB. Interior dots show feature location, whereas circle diameter indicates scale.

In the first set of experiments, the Discrete SURF on FPGA and OpenCV feature detectors were compared for each image in the image set. Both feature detectors were sent to the same OpenCV descriptor and matching stages in order to consistently compare results. In the SURF algorithm, there is no way to limit the detector to an exact number of points. Additionally, because of block RAM limitations on the FPGA, a maximum of 100 features can be saved per image, meaning that the Hessian determinant threshold must be set appropriately high so that the number of features will not exceed the memory. Therefore, because the number of features and matches will vary, the measure of accuracy chosen to evaluate the detector is the number of matches that are correct out of the top fifteen matches produced by the OpenCV matcher. Fifteen matches are used because the experiments have shown that show that fifteen is enough to represent the difference between the detectors. The detected features were rendered as shown in Figure 26 and the matches were rendered as shown in Figure 27 and visually inspected to verify correct matches. The effectiveness of the detector was evaluated in three categories: affine transform, rotation and scale.

Table 1. Discrete SURF and OpenCV feature matching results for images with affine transforms.

	Discrete SURF Detected Features			OpenCV Detected Features		
Affine (degrees)	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct
30	15	5	33%	15	5	33%
20	15	11	73%	15	14	93%
10	15	11	73%	15	12	80%

The results of the FPGA detector for affine transforms produced results that were exactly as expected compared to the OpenCV detector. As shown in Table 1, for angles 0 - 30 the number of correct matches out of the top fifteen matches were almost identical, and even a little better for Discrete SURF over the OpenCV implementation. In Figure 27, it is clear that Discrete SURF detected features produce a better match at the bottom of the image, which has been rotated away from the camera, than the OpenCV detector. In terms of affine transforms in a single direction, the Discrete SURF detector produces outstanding results, which are echoed in the affine transformation matching results in experiment two.

In order to accurately describe and match rotated features for homography estimation, the detector must produce consistent and robust points. As shown by the results in Table 2, the Discrete SURF detector led to about 50% correct matches out

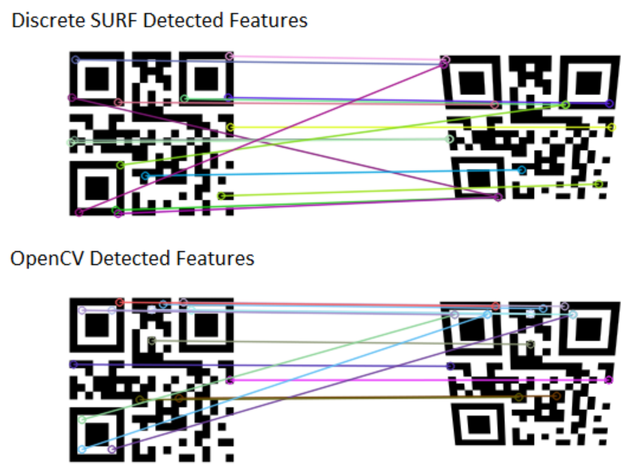


Figure 27. The fifteen best OpenCV matches from Discrete SURF detected points (top) are directly compared to the matches found from OpenCV detected points for images with a 20 degree affine transform.

of the top fifteen matches for every angle compared to around 80% for the OpenCV detector.

Table 2. Discrete SURF and OpenCV feature matching results for rotated images.

	Discrete SURF Detected Features			OpenCV Detected Features		
Rotation (degrees)	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct
90	15	8	53%	15	15	100%
60	15	10	67%	15	12	80%
30	15	10	67%	15	11	73%
15	15	8	53%	15	11	73%

Although the Discrete SURF points performed slightly worse than OpenCV points, 50% matching is sufficient for homography estimation. The main reason that Discrete SURF does not perform as well for feature detection is because it is making roughly 80 detections per frame due to memory constraints, whereas the OpenCV detector has the advantage of capturing around 200 features per frame. Therefore, despite hardware limitations can be easily resolved with a newer FPGA, the Discrete SURF detector produces features that are capable of rotation invariance for homography estimation.

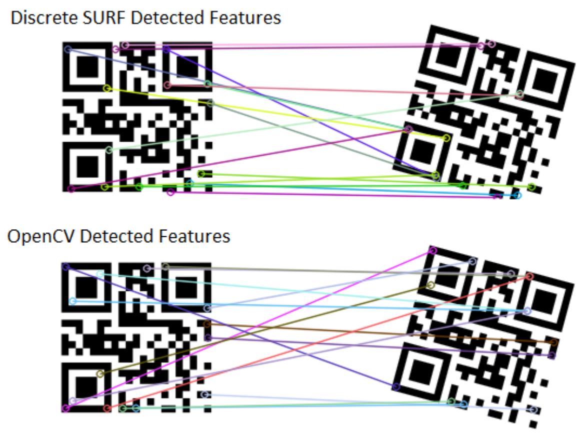


Figure 28. The fifteen best OpenCV matches from Discrete SURF detected points (top) are directly compared to the matches found from OpenCV detected points for 15 degree rotated images.

The matches produced by the Discrete SURF detector for upright images showed around 30% to 80% accuracy for the selected image set. An increase of scale by 125% results in very good matching from the Discrete SURF features compared to OpenCV SURF. The Discrete SURF detector in its current iteration produces enough good matches to provide accurate homography estimates at scales up to 150%. Since this research is focused on homography estimations for UAV approaches, the Discrete SURF algorithm only really needs to provide for invariance as scale increase, which is verified by the above results.

Table 3. Discrete SURF and OpenCV feature matching results for upright images with changes in scale.

	Discrete SURF Detected Features			OpenCV Detected Features		
Scale	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct
150%	15	8	53%	15	11	73%
125%	15	12	80%	15	11	73%
75%	15	4	27%	15	11	73%
50%	4*	3	--	15	13	87%

The smaller scale results did not compare as well against the OpenCV standard which indicates that Discrete SURF scale parameters are not properly adjusted for smaller scale images. For a scale of 50%, the Discrete SURF detector did not detect enough points that were above the Hessian determinant threshold that was used for the rest of the experiments. There are not enough small features to be matched to the decreased scaled image, which is amplified by the effects of the low resolution images being used for these experiments. One potential solution would be to run the algorithm on better hardware which would allow for an increased number of features to be saved per image, and also larger resolutions to be analyzed.

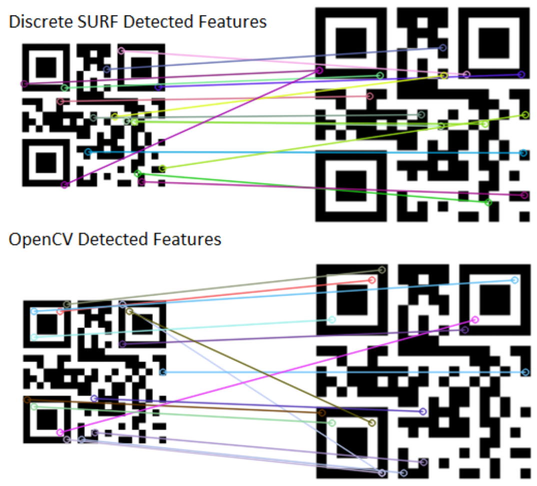


Figure 29. The fifteen best OpenCV matches from Discrete SURF detected points (top) are directly compared to the matches found from OpenCV detected points for 150% scaled images.

Overall, the results of the Discrete SURF feature detector produce detection results that are comparable to the OpenCV detector. For rotation, the Discrete SURF detector produced enough matches to provide for homography estimation. For affine transforms, the Discrete SURF detector produced points that performed nearly exactly as well as the OpenCV implementation. For upright, scaled images, the Discrete SURF detector provided matches that can lead to hompgraphy estimation as the scale of the model is increased during approach.

4.2 Description and Matching Verification Results

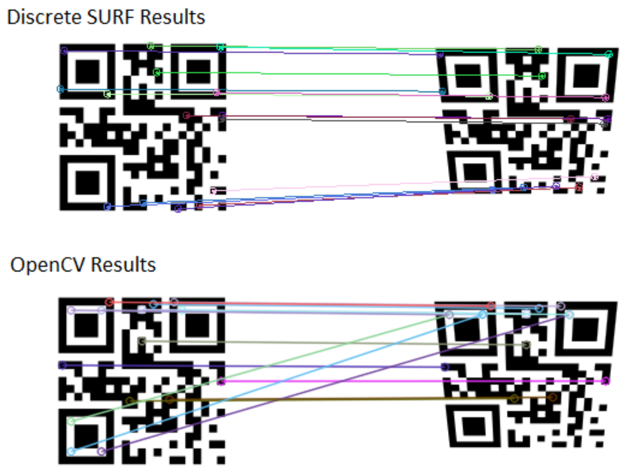


Figure 30. The fifteen best matches by the Discrete SURF algorithm (top) are directly compared to the matches found by the OpenCV SURF (bottom) implementation for images with a 20 degree affine transform.

In experiment two, the entire Discrete SURF implementation was compared directly against the OpenCV SURF implementation for each image in the image set. In order to measure the accuracy between the two implementations, the top fifteen best matches produced are used as in experiment one. The best matches are rendered and visually inspected for both implementations to find correct matches as shown in Figure 31. The effectiveness of the overall implementation is evaluated in three categories: affine transform, rotation and scale. The results of OpenCV SIFT for the same image set are also presented as a reference for the accuracy of Discrete SURF.

The ability to match to an affine transform is essentially a problem of matching different scales at once in the image. The results, shown in Table 4, indicate that Discrete SURF can actually create better affine transform matching depending on its parameters. At 0 - 20 degrees, Discrete SURF has perfect accuracy for the top fifteen matches, whereas the OpenCV implementation falls around 70%. As shown in Figure 30, Discrete SURF achieves its superior performance by being able to match

Table 4. Discrete SURF, OpenCV SURF, OpenCV U-SURF, and OpenCV SIFT feature matching results for images with affine transforms.

	Discrete SURF			OpenCV SURF			OpenCV Upright SURF			OpenCV SIFT		
Affine (degrees)	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct
30	15	12	80%	15	5	33%	15	11	73%	15	4	27%
20	15	15	100%	15	14	93%	15	14	93%	15	5	33%
10	15	15	100%	15	12	80%	15	15	100%	15	9	60%

to the much smaller features in at the portion of the QR code rotated away from the camera. The aerial navigation based inspiration for this research relies heavily on affine transform detection, for which Discrete SURF is well suited.

Table 5. Discrete SURF, OpenCV SURF, OpenCV U-SURF, and OpenCV SIFT feature matching results for rotated images.

	Discrete SURF			OpenCV SURF			OpenCV Upright SURF			OpenCV SIFT		
Rotation (degrees)	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct
90	15	3	20%	15	15	100%	15	0	0%	15	15	100%
60	15	8	53%	15	12	80%	15	0	0%	15	14	93%
30	15	8	53%	15	11	73%	15	3	20%	15	8	53%
15	15	15	100%	15	11	73%	15	14	93%	15	8	53%

For the first time, SURF feature rotation has been implemented in an FPGA. In order to show the improvement over previous, upright-only designs, the results will be compared between the Discrete SURF, OpenCV SURF, and OpenCV U-SURF implementations. The results, shown in Table 5, indicate similar performance between the OpenCV and Discrete SURF implementations. At 15 degrees, there is not enough rotation to change the orientation assignment that happens between images and therefore the matches are largely the same for all three algorithms and Discrete SURF actually outperforms the OpenCV by one correct feature. At 30 through 60 degrees, the accuracy of Discrete SURF is around 50%, which is enough for homography estimation and much improved over U-SURF, which struggles to make any correct matches. At 90 degrees, Discrete SURF makes 20% correct matches, which is again, an improvement over the zero matches made by U-SURF.

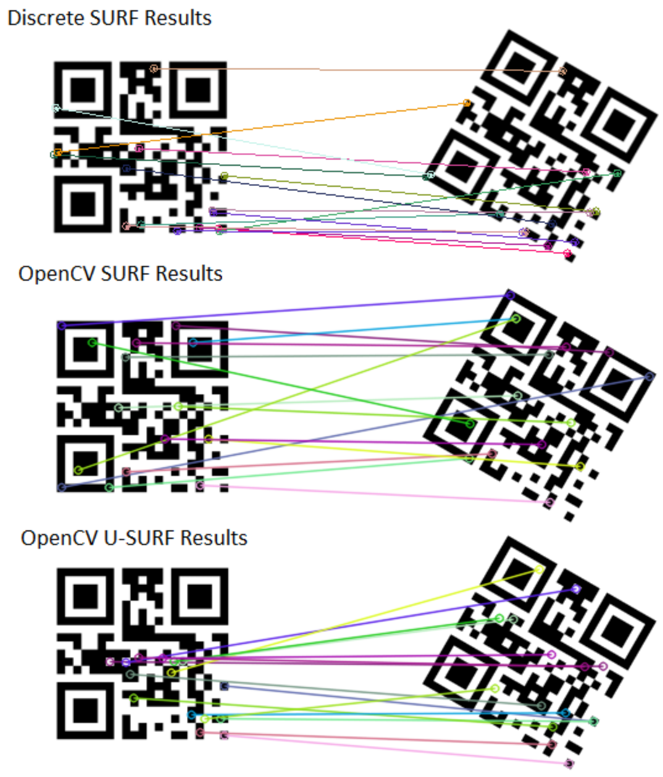


Figure 31. The fifteen best matches by the Discrete SURF algorithm (top) are directly compared to the matches found by the OpenCV SURF (middle) and U-SURF (bottom) implementations for 60 degree rotated images.

Although Discrete SURF is a huge improvement over U-SURF, and therefore the state of the art in FPGA based SURF algorithms, it is not quite at the level of accuracy of the OpenCV SURF algorithm with rotation. The results can again by partially attributed to the smaller number of features that Discrete SURF is able to detect due to RAM limitations. Additionally, because only 24 discrete rotations masks would fit into RAM, it is impossible to precisely calculate the feature angles beyond 15 degree increments, which can lead to some error between rotations.

Overall, the first FPGA SURF with rotation implementation shows matching results that approach the original algorithm and provide significant improvement over the state of the art.

Table 6. Discrete SURF, OpenCV SURF, OpenCV U-SURF, and OpenCV SIFT feature matching results for upright, scaled images.

	Discrete SURF			OpenCV SURF			OpenCV Upright SURF			OpenCV SIFT		
Scale	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct	Number of Matches	Correct Matches	% Correct
150%	15	13	87%	15	11	73%	15	14	93%	15	3	20%
125%	15	15	100%	15	11	73%	15	15	100%	15	6	40%
75%	15	7	47%	15	11	73%	15	15	100%	15	3	20%
50%	4*	1	7%	15	13	87%	15	15	100%	15	6	40%

The results of the scale matching experiment in Table 6 show that the complete Discrete SURF algorithm description and matching stages performed well considering the results of the detector test for scale in the previous experiment. As scale increases, the FPGA implementation performed at or above OpenCV for the top fifteen matches. Additionally, despite using discrete scales for sampling step and feature size, the results aresimilar to the OpenCV implementation.

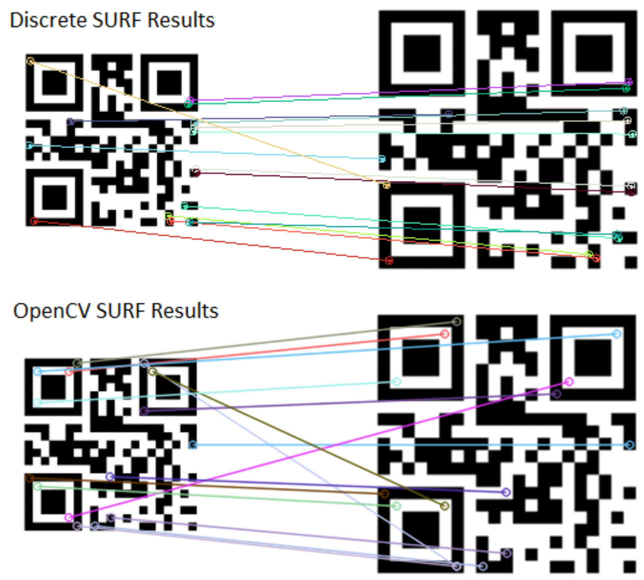


Figure 32. The fifteen best matches by the Discrete SURF algorithm (top) are directly compared to the matches found by the OpenCV SURF (bottom) implementation for 150% scaled images.

For the 50% scaled image, the FPGA was not able to detected enough features to make a comparison which represents the resolution limit of the scale settings chosen for this experiment. The results from experiment one also attribute some of the problems with scale matching to the detector stage.

In order to put these matching results into context for homography estimation, each of the algorithm combinations used in experiments one and two were evaluated to determine whether they met the minimum requirements for homography estimation which is four correct matches. The results are shown in Table 7. Discrete SURF was able to determine a homography for all of the experiments except for 90 degree rotation and 50% scaling for reasons mentioned in the analysis of the previous experiments.

Table 7. Prerequisite check for homography estimation.

		Discrete SURF & OpenCV	Discrete SURF	OpenCV Upright SURF	OpenCV SURF	OpenCV SIFT
affine	30°	✓	✓	✓	✓	✓
	20°	✓	✓	✓	✓	✓
	10°	✓	✓	✓	✓	✓
rotation	90°	✓	✘	✘	✓	✓
	60°	✓	✓	✘	✓	✓
	30°	✓	✓	✘	✓	✓
	15°	✓	✓	✓	✓	✓
scale	150%	✓	✓	✓	✓	✘
	125%	✓	✓	✓	✓	✓
	75%	✓	✓	✓	✓	✘
	50%	✘	✘	✓	✓	✓

4.3 Timing Results

As shown in Table 8, the best case frame rate for this implementation of Discrete SURF on a Virtex-4 FPGA is 19.4fps. For the previous experiments, Discrete SURF was run with one octave of three levels, and a similar number of features for each

Table 8. Best and worst case timing results for different scale runs of the Discrete SURF algorithm.

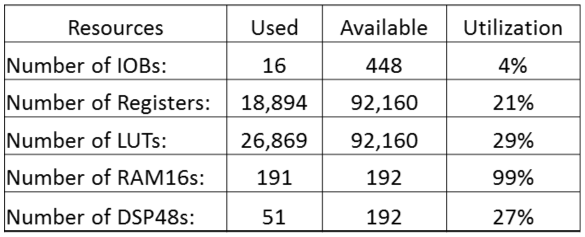
Timing (ms)										
		Integral Image	Detector	Descriptor Best Case	Descriptor Worst Case	Matcher	Total Best Case	Total Worst Case	FPS Best Case	FPS Worst Case
octave 1	Level 1	1.5	35.6	13.5	13.5	0.9	51.5	51.5	19.4	19.4
	Level 2	1.5	71.2	13.5	13.5	0.9	87.1	87.1	11.5	11.5
	Level 3	1.5	106.8	13.5	45.5	0.9	122.7	154.7	8.1	6.5
octave 2	Level 1	1.5	142.4	13.5	45.5	0.9	158.3	190.3	6.3	5.3
	Level 2	1.5	178	13.5	66	0.9	193.9	246.4	5.2	4.1
	Level 3	1.5	213.6	13.5	105.5	0.9	229.5	321.5	4.4	3.1
octave 3	Level 1	1.5	249.2	13.5	66	0.9	265.1	317.6	3.8	3.1
	Level 2	1.5	284.8	13.5	105.5	0.9	300.7	392.7	3.3	2.5
	Level 3	1.5	320.4	13.5	421.5	0.9	336.3	744.3	3.0	1.3

image which result in an estimated frame rate of 6.5 fps. The integral image calculation time is constant for a given image size. The detection stage has a constant runtime for each scale because the detected feature non-maximal suppression feature output are performed in parallel with detection. The description stage can add significant processing time at larger scales, due to the need for individual pixel accesses in calculating the Haar wavelet response which grow with scale. The matcher takes a constant amount of time to compare features and the time shown in the table is the time to compare 50 features from an input image to 50 features from an preloaded image. The largest complication with improving the processing time was fitting the entire design onto the Virtex-4, which is at the time of this research is considered an older FPGA. The size of the design led to large routing delays that severely limited the clock speed. Additionally, had more block RAM been available on the chip, multiple copies of the integral image and captured image could be saved to be accessed simultaneously, which would allow for parallel detection and description.

4.4 FPGA Utilization

As shown in Table 9, the limiting factor for this implementation was the number of block RAMs. In order to fit the entire implementation into FPGA logic in accordance

Table 9. Utilization of the Virtex-4 FPGA by the Discrete SURF algorithm.



Resources	Used	Available	Utilization
Number of IOBs:	16	448	4%
Number of Registers:	18,894	92,160	21%
Number of LUTs:	26,869	92,160	29%
Number of RAM16s:	191	192	99%
Number of DSP48s:	51	192	27%

with one of the original goals of this research, compromises had to be made in terms of speed and accuracy. The large amount of interconnected block RAMs used on the chip led to very high routing delay. Xilinx synthesis tool estimated the maximum clock frequency to be up to 167MHz, however with routing considerations the maximum clock speed was found to be 75MHz. Furthermore, the algorithm could be vastly more parallelized if there was enough RAM to accommodate multiple copies of the integral image and image capture for simultaneous access by the detector and descriptor stages. Finally, the number of matches was limited by the number of features that could be saved in RAM alongside the image data, which may have limited the accuracy of the algorithm in terms of having a feature set that was smaller and less diverse in terms of scale.

4.5 Flight Scenario

The goal of the flight scenario test was to show Discrete SURF's performance in a scenario that fits with the motivation for this research. Figures 33 and 34 show views of the landing pad that can be expected during a real-world land procedure. As shown in Figure 33, the high contrast of the QR code on the platform allows the Hessian threshold to be set high enough to avoid extraneous features detected in the rest of the scene image.

For Figure 34, the image was taken from an approach angle that led to affine, scale, and rotations transformations of the original QR code image. As shown by

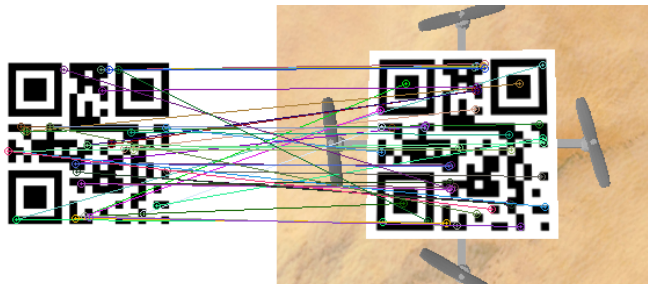


Figure 33. Due to the high contrast of the QR code as a target image, the Hessian determinant threshold is able to be set high enough to avoid unnecessary detection of extraneous scene features.

the detected matches, as well as the OpenCV drawn representation of the calculated homography, Discrete SURF is robust enough to create accurate pose estimations in aerial operations.

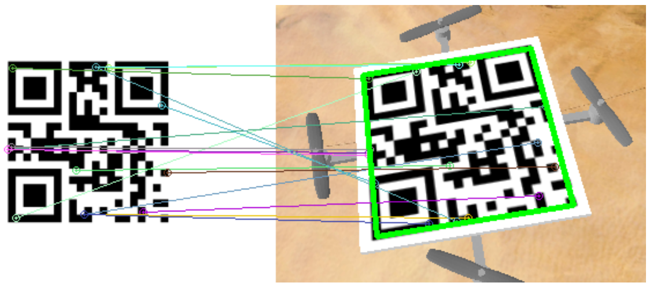


Figure 34. Discrete SURF is able to present the homography estimator with the matches it needs to make an estimation (green box) even at viewing angles that result in a affine, rotated and scaled model.

4.6 Conclusions

The results of the these experiments present and analyze the performance and requirements of Discrete SURF on FPGA. For the first time, SURF with rotation was implemented on an FPGA, showing significant improvement over previous, upright only implementations. The addition of rotation invariance and the performance of Discrete SURF at detecting affine transforms make it well suited for aerial navigation as shown in the scenario test. Furthermore, we achieved increased frames per second over FLEX-SURF despite using inferior hardware[24].

V. Conclusions

This research presents a specialized FPGA implementation of the SURF algorithm, named Discrete SURF, as a low SWAP option for aerial navigation based on homography estimation. In this chapter, conclusions regarding the research and results are discussed as well as the potential for future work.

5.0.1 Contributions.

Discrete SURF is the first FPGA based SURF algorithm that allows for the rotation invariance necessary for aerial navigation. Previous implementations have only explored the use of FPGA SURF for upright images and ground navigation. Additionally, Discrete SURF is the first known FPGA SURF algorithm that is implemented completely in FPGA logic rather than relying on processors for portions of the algorithm. We believe that this logic only implementation will be much easier to port to newer FPGAs with more resources, or onto cameras with FPGAs so that they can directly output matching feature coordinates between frames. Images were processed at 19.4 frames per second for 320x240 pixel resolution with just near the cusp of real-time. Both the frame rate, and image size were limited by the resources on the FPGA, and we believe that if the algorithm were implemented on one of the current generation FPGAs that have nearly ten times the resources, that the results would easily surpass real-time at a much higher resolution and with better results.

5.1 Discussion of Results

The results of the experiments show a mostly complete version of the original SURF algorithm running entirely in FPGA logic. Experiments one and two verify the relative accuracy of the Discrete SURF as compared to the OpenCV benchmark.

Experiment three shows that the algorithm is on the cusp of real-time operation at 19.4 fps, and experiment four shows the potential of Discrete SURF on FPGA as a mounted, low SWAP feature tracker for aerial navigation from homography estimation.

5.1.1 Limitations.

Although the implementation does have increased rotation invariance over Upright SURF, the potential for the same level of accuracy as OpenCV SURF was severely limited by the board resources in a number of ways. Foremost of the board resource limitations was the limited number of block RAMS, which prevented the creation of a greater number of rotation masks to account for more angles, as well as limited the number of features that can be captured. Furthermore, the size of the design was large enough to lead to significant routing delay throughout the system and as a result the system clock and frame rate were both decreased.

5.2 Future Work

Although much progress was made towards FPGA based homography estimation for aerial navigation in this research, there is still a lot of work to be done before Discrete SURF can be used in a UAV system.

5.2.1 Improved Hardware Set.

The first steps taken towards moving Discrete SURF towards real operations is to port it to a current generation FPGA that is better suited for image processing. The Virtex-4, although top of the line in 2005, is not near the current state of the art and an updated FPGA should provide significant performance improvements. The first improvement that should be made after porting to newer hardware is to add additional rotation masks to the description stage in order to get more precise rotation invariance. Additionally, because feature tracking is limited by the resolution of its images, the image resolution should be updated to a modern 1920x1080 for increased distance detection and feature sharpness. For certain future applications, external memory should be used with the Discrete SURF algorithm to acheive performance enhancements at a lower cost than purchasing a more powerful FPGA.

5.2.2 Homography Estimator.

In order to meet the ultimate goal of the research, which is to use this Discrete SURF algorithm for aerial navigation, the next step is to create the homography estimator based on the coordinate pairs produced by the current iteration. Once the homography estimator is included in the system, the FPGA should be interfaced with a flight controller in order to provide position estimation with the hope of real-time flight tests.

5.2.3 Scenario-Based Flight Control.

Although the homography estimation can provide the pose estimation for close distance navigation, it is unsuited for medium to long range navigation. Therefore, in conjunction with the Discrete SURF homography estimator, an aerial navigation platform should also include other monocular position estimation algorithms, such as optical flow, to aid in the full approach. Furthermore, a behavioral architecture that intelligently switches between position estimation algorithms at different stages of approach should be developed to provide more efficient tracking.

5.3 Conclusion

In this research, we deliver a specialized SURF based feature tracking algorithm that is tailored to meet the unique requirements of an FPGA. Our hope is that Discrete SURF will be used in a homography estimation system that provides the robust position estimates needed to facilitate the close-quarters aerial navigation that will be required by future UAVs.

Bibliography

1. E. Ackerman, “UAV battery packs could allow electric planes to fly forever,” *IEEE Spectrum* , July 2012.

2. M. Agrawal, K. Konolige, and R. Bolles, “Localization and mapping for autonomous navigation in outdoor terrains : A stereo vision approach,” in *IEEE Workshop on Applications of Computer Vision, 2007. WACV ’07*, pp. 7–7, Feb. 2007.

3. N. Battezzati, S. Colazzo, M. Maffione, and L. Senepa, “SURF algorithm in FPGA: A novel architecture for high demanding industrial applications,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pp. 161–162, Mar. 2012.

4. H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (SURF),” *Computer Vision and Image Understanding* **110**, pp. 346–359, June 2008.

5. D. Bouris, A. Nikitakis, and I. Papaefstathiou, “Fast and efficient FPGA-based feature detection employing the SURF algorithm,” in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 3–10, May 2010.

6. G. Bradski, “OpenCV,” *Dr. Dobb’s Journal of Software Tools* , 2000.

7. D. Conrad and G. DeSouza, “Homography-based ground plane detection for mobile robot navigation using a Modified EM algorithm,” in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 910–915, May 2010.

8. G. C. H. E. De Croon, H. W. Ho, C. De Wagter, E. Van Kampen, B. Remes, Q. P. Chu, and TU Delft: Aerospace Engineering: Control & Operations, “Optic-flow based slope estimation for autonomous landing,” Sept. 2013.

9. M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM* **24**, pp. 381–395, June 1981.

10. V. Ghadiok, J. Goldin, and W. Ren, “Autonomous indoor aerial gripping using a quadrotor,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4645–4651, Sept. 2011.

11. M. Guerrero, “A comparative study of three image matcing algorithms: Sift, surf, and fast,” *All Graduate Theses and Dissertations* , Jan. 2011.

12. B. Herisse, T. Hamel, R. Mahony, and F.-X. Russotto, “Landing a VTOL unmanned aerial vehicle on a moving platform using optical flow,” *IEEE Transactions on Robotics* **28**, pp. 77–89, Feb. 2012.

13. M. Kaiser, N. Gans, and W. Dixon, “Vision-Based Estimation for Guidance, Navigation, and Control of an Aerial Vehicle,” *IEEE Transactions on Aerospace and Electronic Systems* **46**, pp. 1064–1077, July 2010.

14. D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision* **60**, pp. 91–110, Nov. 2004.

15. B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI’81*, pp. 674–679, Morgan Kaufmann Publishers Inc., (San Francisco, CA, USA), 1981.

16. X. Luo, M. Feuerstein, T. Reichl, T. Kitasaka, and K. Mori, “An Application Driven Comparison of Several Feature Extraction Algorithms in Bronchoscope Tracking During Navigated Bronchoscopy,” in *Medical Imaging and Augmented Reality*, H. Liao, P. J. E. Edwards, X. Pan, Y. Fan, and G.-Z. Yang, eds., *Lecture Notes in Computer Science*(6326), pp. 475–484, Springer Berlin Heidelberg, 2010.

17. R. E. Mandapat, S. Committee, D. Carl, C. Iii, D. J. Duffy, and D. P. M. F. Their, *Development And Evaluation Of Positioning Systems For Autonomous Vehicle Navigation*, 2001.

18. C. Martnez, T. Richardson, P. Thomas, J. L. du Bois, and P. Campoy, “A vision-based strategy for autonomous aerial refueling tasks,” *Robotics and Autonomous Systems* **61**, pp. 876–895, Aug. 2013.

19. V. Murali and S. Birchfield, “Autonomous navigation and mapping using monocular low-resolution grayscale vision,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. CVPRW ’08*, pp. 1–8, June 2008.

20. A. Robertson, “Here are the three things amazon needs to get its delivery drones off the ground.”

21. E. Rosten and T. Drummond, “Fusing points and lines for high performance tracking,” in *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005*, **2**, pp. 1508–1515 Vol. 2, Oct. 2005.

22. J. Schattenberg, T. Lang, S. Batzdorfer, M. Becker, U. Bestmann, and P. Hecker, “Mobile ad-hoc communication in machine swarms for relative positioning based on GNSS-raw data exchange,” in *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pp. 1265–1273, Apr. 2012.

23. B. D. Song, J. Kim, J. Kim, H. Park, J. R. Morrison, and D. H. Shim, “Persistent UAV service: An improved scheduling formulation and prototypes of system components,” *Journal of Intelligent & Robotic Systems* **74**, pp. 221–232, Oct. 2013.

24. J. Svab, T. Krajnik, J. Faigl, and L. Preucil, “FPGA based speeded up robust features,” in *IEEE International Conference on Technologies for Practical Robot Applications, 2009. TePRA 2009*, pp. 35–41, Nov. 2009.

25. T. B. Terriberry, L. M. French, and J. Helmsen, “GPU accelerating speeded-up robust features,” in *Proceedings of 3DPVT*, **8**, pp. 355–362, Citeseer, 2008.

26. M. Trajkovic and M. Hedley, “Fast corner detection,” *Image and Vision Computing* **16**, pp. 75–87, Feb. 1998.

27. N. Ure, G. Chowdhary, T. Toksoz, J. How, M. Vavrina, and J. Vian, “An automated battery management system to enable persistent missions with multiple aerial vehicles,” *IEEE/ASME Transactions on Mechatronics* **20**, pp. 275–286, Feb. 2015.

28. M. Valenti, D. Dale, J. How, D. P. d. Farias, and J. Vian, “Mission health management for 24/7 persistent surveillance operations,” in *AIAA Guidance, Navigation and Control Conference and Exhibit*, American Institute of Aeronautics and Astronautics.

29. K. E. Wenzel, A. Masselli, and A. Zell, “Automatic take off, tracking and landing of a miniature UAV on a moving carrier vehicle,” *Journal of Intelligent & Robotic Systems* **61**, pp. 221–238, Oct. 2010.

30. C. Wu, *SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)*, 2007.

31. J. Zhao, S. Zhu, and X. Huang, “Real-time traffic sign detection using SURF features on FPGA,” in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, Sept. 2013.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 26–03–2015	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Aug 2013 — Mar 2015

4. TITLE AND SUBTITLE FPGA Accelerated Discrete-SURF for Real-Time Homography Estimation	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Leighner, Andrew C., Second Lieutenant, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-MS-15-M-042
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) This block intentionally left blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

This paper describes our hardware accelerated, FPGA implementation of SURF, named Discrete SURF, to support real-time homography estimation for close range aerial navigation. The SURF algorithm provides feature matches between a model and a scene which can be used to find the transformation between the camera and the model. Previous implementations of SURF have partially employed FPGAs to accelerate the feature detection stage of upright only image comparisons. We extend the work of previous implementations by providing an FPGA implementation that allows rotation during image comparisons in order to facilitate aerial navigation. We also expand beyond feature detection as the complete Discrete SURF algorithm is run on the FPGA, rather than piped into processors. This not only minimizes overhead and increases the parallelization of the algorithm, but also allows the algorithm to be easily ported to different FPGAs. Furthermore, the Discrete SURF module is a logic-only implementation that does not rely on external hardware which therefore decreases the overall size, weight and power of the device while also allowing for easy FPGA to ASIC conversion. We evaluate the Discrete SURF algorithm in terms of performance against the original SURF and upright SURF algorithms implemented in OpenCV. Finally, we show how Discrete SURF is more compatible with an aerial navigation scenario than previous works, since rotation invariance must be considered in addition to scale.

15. SUBJECT TERMS

FPGA SURF, feature tracking, relative navigation, homography estimation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Maj John Pecarina, AFIT/ENG
U	U	U	U	80	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-3636, x3368;john.pecarina@afit.edu

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18